Information-Theoretic Privacy in Verifiable Outsourced Computation

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Lucas Schabhüser, M.Sc.

geboren in Münster.

Referenten: Prof. Dr. Johannes Buchmann
Prof. Dr. Peter Ryan

Tag der Einreichung: 13.03.2019
Tag der mündlichen Prüfung: 24.04.2019

Hochschulkennziffer: D 17

Darmstadt 2019

Dissertation von Lucas Scahbhüser:
Information-Theoretic Privacy in Verifiable Outsourced Computation
Technische Universität Darmstadt, Darmstadt, Germany
Tag der mündlichen Prüfung: 24.04.2018
Jahr der Veröffentlichung: 2019

List of Publications

Publications Used in This Thesis

[S1] Denise Demirel, Lucas Schabhüser, and Johannes Buchmann: Privately and Publicly Verifiable Computing Techniques - A Survey. *Springer Briefs in Computer Science*, Springer, 2017. [**Part of Chapter 3**].

[S2] Lucas Schabhüser, Denis Butin, Denise Demirel, and Johannes Buchmann: Function-Dependent Commitments for Verifiable Multi-party Computation. *Information Security - 21st International Conference, ISC 2018*, pages 289–307, Springer, 2018. [**Part of Chapter 4**].

[S3] Patrick Struck, Lucas Schabhüser, Denise Demirel, and Johannes Buchmann: Linearly Homomorphic Authenticated Encryption with Provable Correctness and Public Verifiability. *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017*, pages 142–160, Springer, 2017. [**Part of Chapter 6**].

[S4] Lucas Schabhüser, Patrick Struck, and Johannes Buchmann: A Linearly Homomorphic Signature Scheme from Weaker Assumptions. *Cryptography and Coding - 16th IMA International Conference, IMACC 2017*, pages 261–279, Springer, 2017. [**Part of Chapter 6**].

[S5] Lucas Schabhüser, Denis Butin, and Johannes Buchmann: CHQS: Publicly Verifiable Homomorphic Signatures Beyond the Linear Case. *Information Security Practice and Experience - 14th International Conference, ISPEC 2018*, pages 213–228, Springer, 2018. [**Part of Chapter 5**].

[S6] Lucas Schabhüser, Denis Butin, and Johannes Buchmann: Context Hiding Multi-Key Linearly Homomorphic Authenticators. *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019*, pages 493–513, Springer, 2019. [**Part of Chapter 5**].

[S7] Lucas Schabhüser, Denis Butin, and Johannes Buchmann: Function Depen-

dent Commitments from Homomorphic Authenticators. *Information Security and Privacy - 24th Australasian Conference, ACISP 2019*, pages 399–418, Springer, 2019. [**Part of Chapter 7**].

Other Publications

[S8] Lucas Schabhüser, Denise Demirel, and Johannes Buchmann: An unconditionally hiding auditing procedure for computations over distributed data. *2016 IEEE Conference on Communications and Network Security, CNS 2016*, pages 552–560, IEEE, 2016.

[S9] Matthias Geihs, Lucas Schabhüser, and Johannes Buchmann: Efficient Proactive Secret Sharing for Large Data via Concise Vector Commitments. *Codes, Cryptology and Information Security - Third International Conference, C2SI 2019*, pages 171–194, Springer, 2019.

Abstract

Today, it is common practice to outsource time-consuming computations to the cloud. Using the cloud allows anyone to process large quantities of data without having to invest in the necessary hardware, significantly lowering cost requirements. In this thesis we will consider the following work flow for outsourced computations: A data owner uploads data to a server. The server then computes some function on the data and sends the result to a third entity, which we call verifier.

In this scenario, two fundamental security challenges arise. A malicious server may not perform the computation correctly, leading to an incorrect result. Verifiability allows for the detection of such results. In order for this to be practical, the verification procedure needs to be efficient. The other major challenge is privacy. If sensitive data, for example medical data is processed it is important to prevent unauthorized access to such sensitive information. Particularly sensitive data has to be kept confidential even in the long term.

The field of verifiable computing provides solutions for the first challenge. In this scenario, the verifier can check that the result that was given was computed correctly. However, simultaneously addressing privacy leads to new challenges. In the scenario of outsourced computation, privacy comes in different flavors. One is privacy with respect to the server, where the goal is to prevent the server from learning about the data processed. The other is privacy with respect to the verifier. Without using verifiable computation the verifier obviously has less information about the original data than the data owner - it only knows the output of the computation but not the input to the computation. If this third party verifier however, is given additional cryptographic data to verify the result of the computation, it might use this additional information to learn information about the inputs. To prevent that a different privacy property we call privacy with respect to the verifier is required. Finally, particularly sensitive data has to be kept confidential even in the long term, when computational privacy is not suitable any more. Thus, information-theoretic measures are required. These measures offer protection even against computationally unbounded adversaries.

Two well-known approaches to these challenges are homomorphic commitments and homomorphic authenticators. Homomorphic commitments can provide even

information-theoretic privacy, thus addressing long-term security, but verification is computationally expensive. Homomorphic authenticators on the other hand can provide efficient verification, but do not provide information-theoretic privacy.

This thesis provides solutions to these research challenges – efficient verifiability, input-output privacy and in particular information-theoretic privacy.

We introduce a new classification for privacy properties in verifiable computing. We propose function-dependent commitment, a novel framework which combines the advantages of homomorphic commitments and authenticators with respect to verifiability and privacy. We present several novel homomorphic signature schemes that can be used to solve verifiability and already address privacy with respect to the verifier. In particular we construct one such scheme fine-tailored towards multivariate polynomials of degree two as well as another fine-tailored towards linear functions over multi-sourced data. The latter solution provides efficient verifiability even for computations over data authenticated by different cryptographic keys. Furthermore, we provide transformations for homomorphic signatures that add privacy. We first show how to add computational privacy and later on even information-theoretic privacy. In this way, we turn homomorphic signatures into function-dependent commitments. By applying this transformation to our homomorphic signature schemes we construct verifiable computing schemes with information-theoretic privacy.

Zusammenfassung

Heutzutage ist es üblich aufwendige Berechnungen in der Cloud durchzuführen. Das Nutzen der Cloud erlaubt es jedem große Mengen an Daten zu verarbeiten ohne selbst in die dafür notwendige Hardware investieren zu müssen. Dies führt zu signifikanten Kosteneinsparungen. In dieser Thesis betrachten wir den folgenden Arbeitsablauf für ausgelagerte Berechnungen: Ein Dateneigner lädt seien Daten hoch auf einen Server. Der Server berechnet dann eine Funktion über diesen Daten und sendet das Ergebnis zu einer dritten Partei, welche wir Verifizierer nennen.

In diesem Szenario gibt es zwei fundamentale Sicherheitsprobleme. Ein bösartiger Server kann von der vorgeschriebenen Berechnung abweichen und so ein inkorrektes Ergebnis weitergeben. Effiziente Verifizierbarkeit erlaubt es solche Ergebnisse zu erkennen. Das andere große Sicherheitsproblem ist Vertraulichkeit. Wenn sensible Daten, zum Beispiel Gesundheitsdaten verarbeitet werden, muss sichergestellt werden, dass diese nicht in falsche Hände geraten. Für besonders sensible Daten muss dies sogar langfristig sichergestellt werden. So genanntes Verifiable Computing erlaubt es das erste Problem zu lösen. In diesem Szenario kann der Verifizierer überprüfen ob ein Ergebnis korrekt berechnet wurde. Möchte man zusätzlich Vertraulichkeit gewährleisten, so ergeben sich neue Herausforderungen. Im Szenario von Verifiable Computing sind verschiedene Varianten von Vertraulichkeit zu beachten. Eine ist Vertraulichkeit dem Server gegenüber. Hier geht es darum zu verhindern, dass der Server Informationen über die Daten erhält, welche er verarbeitet. Die andere ist Vertraulichkeit dem Verifizierer gegnüber. Ohne Verifiable Computing hat der Verifizierer offensichtlich weniger Informationen über die Daten als der Dateneigner. Er kennt lediglich das Ergebnis einer Berechnung, nicht aber die Eingabewerte. Wenn ein solcher Verifizierer aber zusätzliche kryptographische Daten erhält um die Korrektheit des Ergebnis zu überprüfen, so kann er versuchen über diese Informationen über die Eingaben abzuleiten. Weiterhin ist es für besonders sensible Daten wichtig, die Vertraulichkeit langfristig zu garantieren. Hierfür reichen Maßnahmen, die auf Annahmen über beschränkte Rechenpower beruhen nicht mehr aus und informationstheoretische Ansätze werden benötigt. Diese erlauben sogar Schutz vor Angreifern welche über unbeschränkte Rechenpower verfügen. Zwei bekannte kryptographische Lösungsansätze für diese Probleme

sind homomorphe Commitment Verfahren und homomorphe Authentikatoren. Homomorphe Commitment Verfahren erlauben sogar informations-theoretisch vertrauliche Verifizierbarkeit, aber die Verifizierung ist extrem aufwendig. Homomorphe Authentikatoren können andererseits effiziente Verifizierung ermöglichen, erlauben aber keine informationstheoretische Vertraulichkeit,

In dieser Arbeit entwickeln wir Lösungen für diese Herausforderungen – effiziente Verifizierbarkeit, Vertraulichkeit für Eingaben und Ausgaben insbesodnere informationstehoretische Vertraulichkeit.

Wir beschreiben zunächst eine neue Klassifizierung für die verschiedenen Vertraulichkeitsaspekte im Verifiable Computing. Dann präsentieren wir "Function-Dependent Commitment" Verfahren, welche die Vorteile von homomorphen Authentikatoren und Commitment Verfahren vereinen. Wir zeigen weiterhin, wie man homomorphe Authentikatoren transformieren kann um zusätzliche Vertaulichkeitsgarantien zu erhalten. Auf diese Art können wir homomorphe Authentikatoren in "Function-Dependent Commitment" Verfahren überführen. Indem wir dies auf die in dieser Arbeit entwickelten homomorphen Authentikatoren anwenden erhalten wir Verifiable Computing Verfahren mit informationshteoretischer Vertraulichkeit.

Contents

List of Tables

1 Introduction

Motivation

Today, it is common practice to outsource time-consuming computations to the cloud. Any data owner can upload its data to a cloud server and asks the server to perform a computation on these data. This allows anyone to process large quantities of data without having to own and maintain the necessary hardware, leading to significantly reduced costs. In this thesis we will consider the following scenario: A (possibly resource-constrained) *data owner* uploads data to some *server* (or even multiple servers) and asks the server to perform a computation on these data. The server then performs this computation and provides the result either to the data owner or to a third party. For example, statistics on health data can be given to an insurance company. In this scenario, two fundamental security challenges arise.

The first is *verifiability*, that is the ability to detect if the server executed a given computation correctly. Here we have a *verifier* receiving the result. Note that the verifier and the data owner may be the same entity. However, we will consider the more generic scenario of a third party verifier. If the verifier is the data owner it could in theory just redo the computation and thus detect any incorrect result. This, however, makes outsourcing pointless. If the result is *efficiently verifiable*, the verifier can check the correctness in less time than the computation itself requires. If the verifier is not the data owner, it cannot check the correctness by doing a re-computation, since it does not have the input data, and requires some other means - *public verifiability*.

The second fundamental challenge is *privacy*. In the scenario of outsourced computation, this comes in different flavors. One is privacy with respect to the server, where the goal is to prevent the server from learning about the processed data. This applies both to inputs and outputs to the computation. The other is privacy with respect to the verifier. A verifier who is given a result of a computation obviously does not have all information about the inputs to this computation. If additional cryptographic data are provided to make this result verifiable, this should not leak any more information than can be derived from the result. Likewise, this

type of privacy can be defined for both inputs and outputs. Note that this is particularly challenging if we do not have a single data owner but inputs to the computation are provided by multiple data owners, as any cryptographic data provided by them is produced under different keys. This scenario for example arises, when computing the averages on health data provided by multiple patients. Particularly sensitive data, e.g. medical data, must remain private even in the long term. Thus, privacy that depends on the computational hardness of certain problems is insufficient. Here information-theoretic privacy is required.

In this thesis, we address the challenge of providing information-theoretic privacy in efficiently verifiable outsourced computations.

There already exist partial solutions to this problem. One of the building blocks to construct verifiable computing schemes are homomorphic authenticators [12]. The general idea of homomorphic authenticators is the following. Before delegating inputs to a function, the input values are authenticated. The homomorphic property allows the server to compute an authenticator to the output of a given function from the authenticators to the inputs to said function. In the public key setting, homomorphic authenticators are called *homomorphic signatures*. In the private key setting, they are called *homomorphic MACs*.

In order to achieve information-theoretic privacy with respect to the server, we first require a way to store the data in a long term secure way. A well known approach to this is proactive secret sharing (see e.g. [33]). It is furthermore known that multi-party computation is possible based on any linear secret sharing scheme (see e.g. [51]). Therefore it is even possible to perform computations on data that is stored in an information-theoretically secure fashion. Another fundamental building block for long-term security are commitments (see e.g. [32]). Homomorphic commitments can also be used to achieve verifiability (see e.g. [54]), though in general without the property of efficient verification.

As discussed above there are multiple types of privacy to be considered. So far no verifiable computing scheme offers *all* the privacy properties we require. In particular no scheme offers *complete information-theoretic privacy*, i.e. information-theoretic input-output privacy with respect to both verifier and server. In this thesis, we provide the first such solutions.

Contribution and Outline

The following paragraphs will now summarize our contributions and elaborate the structure of this thesis.

Background (Chapter 2).

First, we explain the relevant background. This includes the basic definitions for homomorphic authenticators. Furthermore, we briefly introduce secret sharing and commitment schemes. Subsequently, we provide the basic definitions for verifiable computing. Finally, we present the cryptographic hardness assumptions used within this thesis.

Classifying Verifiable Computing Schemes by Their Privacy Properties (Chapter 3).

In this chapter, we provide a detailed classification for privacy in verifiable computing, differentiating between four types of privacy:

IPS: input privacy with respect to the server
OPS: output privacy with respect to the server
IPV: input privacy with respect to the verifier
OPV: output privacy with respect to the verifier

Input and output privacy with respect to the server enable the outsourcing of computations over sensitive data even to an untrustworthy server. Note that we implicitly include the case of multiple servers. Input privacy with respect to the verifier guarantees that inputs to a computation are not revealed. Since we consider the case of publicly verifiable computations, and thus anyone can potentially have access to the verification data, this scenario cannot be solved by the client's choice of trusted servers. Output privacy with respect to the verifier enables the modular use of verifiable computing schemes, i.e. checking for the correctness of some intermediate result without needing to know said result. Furthermore both input and output privacy with respect to the verifier are necessary properties to achieve input and output privacy with respect to the servers for publicly verifiable schemes, as anyone, so in particular the server has access to the verification data. If a verifiable computing scheme achieves all four types of privacy, we say it achieves *complete privacy*. We will use this classification of privacy properties to compare existing verifiable computing schemes.

Function-Dependent Commitment Schemes (Chapter 4).

Having seen that no prior schemes achieves all four privacy properties identified in Chapter 3, we now focus on constructing the first efficiently verifiable computing scheme for linear functions that provides complete privacy, even in an information-theoretic sense. In order to achieve this we present the fundamental building block of our information-theoretically private schemes — *function-dependent commitments (FDC)*. This notion is a generalization of both homomorphic authenticators and

commitments. On a high level we first present the framework of FDCs and provide definitions of their properties. We then provide the first instantiation of a FDC scheme, supporting linear functions. Finally, we show how our instantiation can be combined with a linear secret sharing scheme to build the first efficiently verifiable computing scheme that provides complete privacy.

Context Hiding Homomorphic Authenticators (Chapter 5).

In this chapter, we present two novel homomorphic authenticator schemes that each satisfy information-theoretic IPV (introduced in Chapter 3). These new schemes will be the basis for further verifiable computing schemes achieving complete privacy. We start by introducing the first publicly verifiable homomorphic authenticator scheme fine-tailored for quadratic functions based on bilinear maps. We then present the first multi-key linearly homomorphic authenticator scheme to achieve input privacy with respect to the verifier. Such a scheme can be used to verify the results of a computation over inputs from *different data owners*.

Adding Computational Privacy to Homomorphic Authenticators (Chapter 6).

In this chapter, we show how to add computational IPS and OPS to linearly homomorphic authenticators. We improve on the work of Catalano et al. [43]. They presented a generic transformation for linearly homomorphic authenticator schemes. However, prior to this work no instantiations were known, that did not suffer from false negatives – correct results identified as incorrect by the verifiable computing scheme. We present the first instantiations that avoid this problem. One construction is based on the RSA problem and one on the CDH problem.

Adding Information-Theoretic Privacy to Homomorphic Authenticators (Chapter 7).

In this chapter, we present a transformation for homomorphic authenticators that adds information-theoretic privacy properties. This transformation adds OPV, IPS and OPS to homomorphic authenticators. To this end, we investigate the relation between homomorphic authenticators and the FDCs introduced by us in Chapter 4. We both show how to transform any FDC into a homomorphic authenticator as well as how homomorphic authenticators with specific properties can be combined with commitments to obtain FDCs. Finally we apply this transformation to the two schemes introduced in Chapter 5. These schemes already achieved IPV. Applying our transformation results in two verifiable computing schemes with complete information-theoretic privacy.

Conclusion (Chapter 8).

This chapter concludes this thesis with a summary of results and a discussion on possible directions of future research.

2 Background

This chapter contains revised and extended parts of the background sections published in [S1], [S2], [S3], [S4], [S5], [S6], and [S7].

2.1 Labeled Programs

To accurately describe both correct and legitimate operations for homomorphic authenticators, we use *multi-labeled programs* similarly to Backes, Fiore, and Reischuk [12]. The basic idea is to append a function by several identifiers, in our case *input identifiers* and *dataset identifiers*. Input identifiers label in which order the input values are to be used and dataset identifiers determine which authenticators can be homomorphically combined. The idea is that only authenticators created under the same dataset identifier can be combined. We now give formal definitions.

A *labeled program* $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_n)$, where $f : \mathcal{M}^n \to \mathcal{M}$ is a function with n inputs and $\tau_i \in \mathcal{T}$ is a label for the i^{th} input of f from some set $\mathcal{T}$. Given a set of labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_N$ and a function $g : \mathcal{M}^N \to \mathcal{M}$, they can be composed by evaluating g over the labeled programs, i.e. $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. Note that this is an abuse of notation analogous to function composition. The identity program with label τ is given by $\mathcal{I}_\tau = (f_{id}, \tau)$, where $f_{id} : \mathcal{M} \to \mathcal{M}$ is the identity function. The program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ of the labeled program $\mathcal{P}$ and a dataset identifier Δ. Given a set of k multi-labeled programs with the same dataset identifier Δ, i.e. $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_N, \Delta)$, and a function $g : \mathcal{M}^N \to \mathcal{M}$, a composed multi-labeled program $\mathcal{P}^*_\Delta$ can be computed, consisting of the pair $(\mathcal{P}^*, \Delta)$, where $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. Analogously to the identity program for labeled programs, we refer to a multi-labeled identity program by $\mathcal{I}_{(\tau,\Delta)} = ((f_{id}, \tau), \Delta)$.

Definition 2.1 (Well Defined Program [37])**.** *A labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is* well defined *with respect to a list $L \subset \mathcal{T} \times \mathcal{M}$ if one of the two following cases holds: First, there are messages $m_1, \ldots, m_n$ such that $(\tau_i, m_i) \in L \; \forall i \in [n]$.*

Second, there is an $i \in [n]$ *such that* $(\tau_i, \cdot) \notin L$ *and* $f(\{m_j\}_{(\tau_j,m_j)\in L} \cup \{m'_k\}_{(\tau_k,\cdot)\notin L})$ *is constant over all possible choices of* $m'_k \in \mathcal{M}$.

If f is a linear function, the labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, with $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$ fulfills the second condition if and only if $f_k = 0$ for all $k \in [n]$ such that $(\tau_k, \cdot) \notin L$.
Freeman pointed out [60] that it may generally not be possible to decide whether a multi-labeled program is well defined with regard to a list L. For this, we use the following Lemma:

Lemma 2.2 ([35])**.** *Let* $\lambda, n, d \in \mathbb{N}$ *and let* F *be the class of arithmetic circuits* $f : \mathbb{F}^n \to \mathbb{F}$ *over a finite field* $\mathbb{F}$ *of order* p*, such that the degree of* f *is at most* d*, for* $\frac{d}{p} \leq \frac{1}{2}$*. Then, there exists a probabilistic polynomial time (PPT) algorithm that for any given* $f \in F$*, decides if there exists* $y \in \mathbb{F}$*, such that* $f(u) = y$ *for all* $u \in \mathbb{F}$ *(i.e. if* f *is constant) and is correct with probability at least* $1 - 2^{-\lambda}$.

2.2 Homomorphic Authenticators

Definition 2.3 (Homomorphic Authenticator [59](adapted))**.** *A homomorphic authenticator scheme* HAuth *is a tuple of the following probabilistic polynomial time algorithms:*

- $\mathsf{Setup}(1^\lambda)$: *On input a security parameter* λ*, the algorithm returns a set of public parameter* pp*, consisting of (at least) the description of an identifier space* $\mathcal{T}$*, a message space* $\mathcal{M}$*, and a set of admissible functions* $\mathcal{F}$*. The public parameters* pp *will implicitly be inputs to all following algorithms even if not explicitly specified.*
- $\mathsf{KeyGen}(\mathsf{pp})$: *On input the public parameters* pp*, the algorithm returns a key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*, where* sk *is the secret key authentication key,* ek *is a public evaluation key, and* vk *is a verification key that can be either private or public.*
- $\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$: *On input a secret key* sk*, a dataset identifier* Δ*, a label* τ*, and a message* m*, the algorithm returns an authenticator* σ.
- $\mathsf{Eval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$: *On input a function* $f : \mathcal{M}^n \to \mathcal{M}$ *and a set* $\{\sigma_i\}_{i\in[n]}$ *of authenticators, and an evaluation key* ek*, the algorithm returns an authenticator* σ.
- $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$*, a verification key* vk*, a message* $m \in \mathcal{M}$*, and an authenticator* σ *, the algorithm returns either '1'(accept), or '0'(reject).*

This definition is adapted from [59] to the single key setting. Compared to earlier definitions e.g. [37], key generation is split up into a setup of public parameters and the key generation itself. Note that, unlike previous definitions, our definition does

not explicitly demand the scheme to require several properties, instead providing explicit definitions for the respective properties later on. In particular we do not demand succinctness as an integral property of homomorphic authenticators. In this thesis we particularly aim for efficient verification (see Def. 2.8) and context hiding public verifiability, which is not an integral property in earlier definitions of homomorphic authenticators. Therefore a scheme achieving efficient verification and/or context hiding but not succinctness is still of interest.

If vk is private, we call HAuth a *homomorphic MAC*, while for a public vk we call it a *homomorphic signature*.

This described the case for a single key homomorphic authenticator. This can naturally be extended to multi-key homomorphic authenticators. Note that in the multi-key case labels consist of two parts, an *identity* id and an input identifier τ. In the single key case there is only the input identifier τ. In the multi-key scenario we will call labels $l = (\mathsf{id}, \tau)$ to distinguish them.

Definition 2.4 (Multi-Key Homomorphic Authenticator [59](adapted))**.** *A multi-key homomorphic authenticator scheme* $\mathsf{MKHAuth}$ *is a tuple of the following probabilistic polynomial time (PPT) algorithms:*

- $\mathsf{Setup}(1^\lambda)$: *On input a security parameter* λ*, the algorithm returns a set of public parameter* pp*, consisting of (at least) the description of an identifier space* $\mathcal{T}$*, an identity space* ID*, a message space* $\mathcal{M}$*, and a set of admissible functions* $\mathcal{F}$*. Given* $\mathcal{T}$ *and* ID *the label space of the scheme is defined as* $\mathcal{L} = \mathsf{ID} \times \mathcal{T}$*. The public parameters* pp *will implicitly be inputs to all following algorithms even if not explicitly specified.*
- $\mathsf{KeyGen}(\mathsf{pp})$: *On input the public parameters* pp*, the algorithm returns a key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*, where* sk *is the secret key authentication key,* ek *is a public evaluation key, and* vk *is a verification key that can be either private or public.*
- $\mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$: *On input a secret key* sk*, a dataset identifier* Δ*, a label* $l = (\mathsf{id}, \tau)$*, and a message* m*, the algorithm returns an authenticator* σ*.*
- $\mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$: *On input a function* $f : \mathcal{M}^n \to \mathcal{M}$ *and a set* $\{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]}$ *of authenticators and evaluation keys, the algorithm returns an authenticator* σ*.*
- $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$*, a set of verification key* $\{\mathsf{vk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}}$*, corresponding to the identities* id *involved in the program* $\mathcal{P}$*, a message* $m \in \mathcal{M}$*, and an authenticator* σ *, the algorithm returns either '1'(accept), or '0'(reject).*

To avoid an overhead in notation we will use the single key setting in the rest of this section unless noted otherwise.

We now define properties relevant for the analysis of homomorphic authenticator schemes: authentication correctness, evaluation correctness, succinctness, unforgeability, efficient verification and context hiding.

Correctness naturally comes in two forms. We require both authenticators created directly with a secret signing key as well as those derived by the homomorphic property to verify correctly.

Definition 2.5 (Authentication Correctness [41])**.** *A homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *satisfies* authentication correctness *if, for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, any label* $\tau \in \mathcal{T}$*, any dataset identifier* $\Delta \in \{0,1\}^*$*, any message* $m \in \mathcal{M}$*, and any authenticator* $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$ *we have* $\mathsf{Ver}(\mathcal{I}_{(\tau,\Delta)}, \mathsf{vk}, m, \sigma) = `1'$*, where* $\mathcal{I}_{\tau,\Delta}$ *is the multi-labeled identity program.*

Definition 2.6 (Evaluation Correctness [41])**.** *A homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *satisfies* evaluation correctness *if, for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, for any dataset identifier* $\Delta \in \{0,1\}^*$*, and any set of program/message/authenticator triples* $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$*, such that* $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ *the following holds: Let* $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$*, and* $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, \{\sigma_i\}_{i\in[N]})$*. Then* $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, m^*, \sigma^*) = 1$ *holds.*

We now consider two properties impacting the practicality of homomorphic authenticator schemes. Succinctness on a high level guarantees that bandwidth requirements for deploying such a scheme are low. Efficient verification allows for low computational effort on behalf of the verifier.

Definition 2.7 (Succinctness [59])**.** *A homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *is said to be succinct if the size of every authenticator depends only logarithmically on the size of a dataset. More formally, let* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*,* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, and* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, \tau_i, m_i)$ *for all* $i \in [n]$*. A homomorphic authenticator is said to be succinct if there exists a fixed polynomial* p *such that* $|\sigma| = p(\lambda, \log n)$*, where* $\sigma = \mathsf{Eval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$*. However, we allow authenticators to depend on the number of keys involved in the computation (in the multi-key setting). A multi-key homomorphic authenticator is said to be succinct if there exists a fixed polynomial* p *such that* $|\sigma| = p(\lambda, k, \log n)$*, where* $\sigma = \mathsf{Eval}(f, \{\sigma_i, \mathsf{ek}_{\mathsf{id}_i}\}_{i\in[n]})$ *and* $k = |\{\mathsf{id} \in \mathcal{P}\}|$*.*

Like Libert and Yung [80], we call a key *concise* if its size is independent of the input size n.

Definition 2.8 (Efficient Verification [37])**.** *A homomorphic authenticator scheme for multi-labeled programs allows for* efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$: *Given a labeled program* $\mathcal{P} = (f, l_1, \ldots, l_n)$*, and verification key* vk *this algorithm generates a concise verification key* $\mathsf{vk}_\mathcal{P}$*. This does* not *depend on a dataset identifier* Δ*.*

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$*: Given a concise verification key* $\mathsf{vk}_{\mathcal{P}}$*, a dataset* Δ*, a message* m*, and an authenticator* σ*, it outputs '1' or '0'.*

The above algorithms are required to satisfy the following two properties:

Correctness: *Let* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ *be an honestly generated key triple and* $(\mathcal{P}_\Delta, m, \sigma)$ *be a tuple. Then, for every* $\mathsf{vk}_{\mathcal{P}} \xleftarrow{\$} \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$*, we have*

$$\Pr[\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma) \neq \mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)] = \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ *denotes any function negligible in the security parameter* λ*.*

Amortized Efficiency: *Let* $\mathcal{P}$ *be a program, let* $m_1, \ldots, m_n$ *be valid input values and let* $t(n)$ *be the time required to compute* $\mathcal{P}(m_1, \ldots, m_n)$ *with output* m*. Then, for any* $\mathsf{vk}_{\mathcal{P}} \xleftarrow{\$} \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$*, and any* $\Delta \in \{0,1\}^*$ *the time required to compute* $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$ *is* $t' = o(t(n))$*, where* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, l_i, m_i)$ *for* $i \in [n]$*, and* $\sigma \leftarrow \mathsf{Eval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek}))$*.*

Here, *efficiency* is used in an amortized sense. There is a function-dependent pre-processing phase, so that verification cost amortizes over multiple datasets.

For the notion of unforgeability of a homomorphic authenticator scheme (Setup, KeyGen, Auth, Eval, Ver), we define the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. During the experiment, the adversary $\mathcal{A}$ can adaptively query the challenger $\mathcal{C}$ for authenticators on messages of his choice under labels of his choice. He can also make verification queries. Intuitively, the homomorphic property allows anyone (with access to the evaluation keys) to derive new authenticators. This can be checked by the use of the corresponding program in the verification algorithm. An adversary should however not be able to derive authenticators beyond that.

Definition 2.9 ($\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ [41])**.**

Setup: $\mathcal{C}$ *runs* $\mathsf{Setup}(1^\lambda)$ *to obtain the public parameters* pp *that are sent to* $\mathcal{A}$*, runs* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and gives* ek *to* $\mathcal{A}$*.*

Authentication Queries: $\mathcal{A}$ *can adaptively submit queries of the form* (Δ, τ, m) *where* Δ *is a dataset identifier,* $\tau \in \mathcal{T}$ *is a label, and* $m \in \mathcal{M}$ *is a message of its choice.* $\mathcal{C}$ *answers as follows:*

- *If* (Δ, τ, m) *is the first query for the dataset* Δ*,* $\mathcal{C}$ *initializes an empty list* $L_\Delta = \varnothing$ *and proceeds as follows.*
- *If* (Δ, τ, m) *is such that* $(\tau, m) \notin L_\Delta$*,* $\mathcal{C}$ *computes* $\sigma_\tau \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$ *, returns* σ_τ *to* $\mathcal{A}$ *and updates the list* $L_\Delta \leftarrow L_\Delta \cup (\tau, m)$*.*
- *If* (Δ, τ, m) *is such that* $(\tau, \cdot) \in L_\Delta$ *(which means that the adversary had already made a query* (Δ, τ, m') *f), then* $\mathcal{C}$ *ignores the query.*

Verification Queries: *$\mathcal{A}$ is also given access to a verification oracle. Namely the adversary can submit a query $(\mathcal{P}_\Delta, m, \sigma)$ and $\mathcal{C}$ replies with the output of $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$.*

Corruption Queries(only in the multi-key setting): *The adversary $\mathcal{A}$ has access to a corruption oracle. At the beginning of the experiment, the challenger $\mathcal{C}$ initializes an empty list $L_{\mathsf{corr}} = \varnothing$ of corrupted identities. During the game $\mathcal{A}$ can adaptively query identities $\mathsf{id} \in \mathsf{ID}$. If $\mathsf{id} \notin L_{\mathsf{corr}}$ then $\mathcal{C}$ replies with the triple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ (that is generated using KeyGen if not done before) and updates the list $L_{\mathsf{corr}} \leftarrow L_{\mathsf{corr}} \cup \mathsf{id}$. If $\mathsf{id} \in L_{\mathsf{corr}}$, then $\mathcal{C}$ replies with the triple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ assigned to id before.*

Forgery: *In the end, $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. The experiment outputs '1' if the tuple returned by $\mathcal{A}$ is a forgery as defined below (see Def. 2.10), and '0' otherwise.*

This describes the case of privately verifiable homomorphic authenticators. For homomorphic signatures vk is given to the adversary.

Definition 2.10 (Forgery [37])**.** *Consider a run of $\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ where $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is the tuple returned by the adversary in the end of the experiment, with $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$. This is a forgery if $\mathsf{Ver}(\mathcal{P}^*{}_{\Delta^*}, \mathsf{vk}, m^*, \sigma^*) = 1$, and at least one of the following properties is satisfied:*

Type 1 Forgery: *The list L_{Δ^*} was not initialized during the security experiment, i.e. no message was ever committed under the dataset identifier Δ^*.*

Type 2 Forgery: *$\mathcal{P}^*{}_{\Delta^*}$ is well defined with respect to list L_{Δ^*} and m^* is not the correct output of the computation, i.e. $m^* \neq f^*(m_1, \ldots, m_n)$*

Type 3 Forgery: *$\mathcal{P}^*{}_{\Delta^*}$ is not well defined with respect to L_{Δ^*} (see Def. 2.1).*

Definition 2.11 (Unforgeability [37])**.** *A homomorphic authenticator scheme* HAuth *is* unforgeable *if for any PPT adversary $\mathcal{A}$ we have*

$$\Pr[\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

We will now provide a weaker version of unforgeability. In the following experiment $\mathsf{weak} - \mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$, the adversary has to declare the message components of the later signing queries before the key generation and can later on specify in which dataset Δ_j it wants to query it.

Definition 2.12 ($\mathsf{weak} - \mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ [41])**.**

Declaration of Messages *$\mathcal{A}$ outputs a list of possible messages $\{m_{\tau,j}\}_{\tau\in L, j=1}^{Q} \subset \mathcal{M}$ where Q is the number of datasets to be queried.*

Setup: *$\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters pp that are sent to $\mathcal{A}$, runs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and gives ek to $\mathcal{A}$.*

Authentication Queries: *$\mathcal{A}$ can adaptively submit queries of the form $(\Delta_j, \tau, m_{\tau,j})$ where Δ_j is a dataset, $\tau \in \mathcal{T}$ is a label, and $m_{\tau,j} \in \mathcal{M}$ is a declared message. $\mathcal{C}$ answers as follows:*

If $(\Delta_j, \tau, m_{\tau,j})$ is the first query with dataset identifier Δ_j, $\mathcal{C}$ initializes an empty list $L_{\Delta_j} = \varnothing$ and proceeds as follows.
If (Δ, τ, m) is such that $(\tau, m) \notin L_\Delta$, $\mathcal{C}$ computes $\sigma_\tau \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$, returns σ_τ to $\mathcal{A}$ and updates the list $L_\Delta \leftarrow L_\Delta \cup (\tau, m)$.
If (Δ, τ, m) is such that $(\tau, \cdot) \in L_\Delta$ (which means that the adversary had already made a query (Δ, τ, m') f), then $\mathcal{C}$ ignores the query.

Verification Queries: *$\mathcal{A}$ is also given access to a verification oracle. Namely the adversary can submit a query $(\mathcal{P}_\Delta, m, \sigma)$ and $\mathcal{C}$ replies with the output of $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$.*

Forgery: *In the end, $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. The experiment outputs '1' if the tuple returned by $\mathcal{A}$ is a forgery as defined above (see Def. 2.10), and '0' otherwise.*

Definition 2.13 (Weak Unforgeability [41])**.** *A homomorphic authenticator scheme* HAuth *is* weakly-unforgeable *if for any PPT adversary $\mathcal{A}$ we have*

$$\Pr[\mathsf{weak} - \mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

Theorem 2.14. *If* HAuth *is a weakly-unforgeable homomorphic signature scheme in the sense of Def. 2.13 then it can be transformed into an unforgeable homomorphic signature scheme in the sense of Def. 2.11.*

Proof. This is a direct corollary of [41, Theorem 1]. □

Moreover the computational assumptions on which the weak unforgeability is based on are preserved by this transformation.

In the case of multi-key homomorphic authenticators we will also consider a relaxation of the unforgeability definition in which the adversaries ask for corruptions in a non-adaptive way. More precisely, we say that an adversary $\mathcal{A}$ makes non-adaptive corruption queries if for every identity id asked to the corruption oracle, id was not queried earlier in the game to the authentication oracle or the verification oracle. For this class of adversaries, corruption queries are of no help as the adversary can generate keys on its own. We will use the following Proposition:

Proposition 2.15 ([59, Proposition 1])**.** MKHAuth *is unforgeable against adversaries that do not make corruption queries if and only if* MKHAuth *is unforgeable against adversaries that make non-adaptive corruption queries.*

Additionally we will make use of the following statements.

Proposition 2.16 ([60, Proposition 2.3])**.** *Let* HAuth = (Setup, KeyGen, Auth, Eval, Ver) *be a linearly homomorphic signature scheme over a message space* $\mathcal{M} \subset R^T$ *for some ring* R*. If* HAuth *is secure against Type 2 forgeries, then* HAuth *is also secure against Type 3 forgeries.*

Proposition 2.17 ([41, Proposition 2])**.** *Let* $\lambda \in \mathbb{N}$ *be the security parameter, and let* $\mathcal{F}$ *be the class of arithmetic circuits* $f : \mathbb{F}^n \to \mathbb{F}$ *over a finite field* $\mathbb{F}$ *of order* p *and such that the degree of* f *is at most* d*, for* $\frac{d}{p} < \frac{1}{2}$*. Let* HAuth *be a homomorphic signature with message space* $\mathbb{F}$ *and let* $\mathcal{E}_b$ *be the event that the adversary returns a Type-*b *forgery (for* $b = 1, 2, 3$*) (see Def. 2.10) in Experiment 2.9. Then, if for any adversary* $\mathcal{A}$ *we have* $\Pr[\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1 \ \wedge \ \mathcal{E}_2] \leq \epsilon$*, then for any adversary* $\mathcal{A}'$ *producing a Type 3 forgery it holds that* $\Pr[\mathsf{HomUF-CMA}_{\mathcal{A}',\mathsf{HAuth}}(\lambda) = 1 \ \wedge \ \mathcal{E}_3] \leq \epsilon + 2^{-\lambda}$*.*

We are now ready to provide our notion of input privacy, in the form of the context hiding property.

Definition 2.18 (Context Hiding [37])**.** *A homomorphic authenticator scheme for multi-labeled programs is* context hiding *if there exist two additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ *and* $\mathsf{HideVer}(\mathsf{vk}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ *such that:*

Correctness: *For any* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and any tuple* $(\mathcal{P}_\Delta, m, \sigma)$*, such that* $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma) = 1$*,*
and $\tilde{\sigma} \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$*, it holds that* $\mathsf{HideVer}(\mathsf{vk}, \mathcal{P}_\Delta, m, \tilde{\sigma}) = 1$*.*

Unforgeability: *The homomorphic authenticator scheme is unforgeable in the sense of Def. 2.11 when replacing the algorithm* Ver *with* HideVer *in the security experiment.*

Context Hiding Security: *There is a simulator* Sim *such that, for any fixed (worst-case) choice of* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, any multi-labeled program* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$*, messages* $m_1, \ldots, m_n$*, and distinguisher* $\mathcal{D}$ *there exists a function* $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ *such that*

$$|\Pr[\mathcal{D}(\mathcal{I}, \mathsf{Hide}(\mathsf{vk}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathcal{I}, \mathsf{Sim}(\mathcal{I}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda),$$

where $\mathcal{I} = (\mathsf{sk}, (m_\tau, \sigma_\tau), \ \sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \ \tau_i, m_i), \ m \leftarrow f(m_1, \ldots, m_n),$ $\sigma \leftarrow \mathsf{Eval}(f, \{\sigma_i\}_{i \in [n]})$*, and the probabilities are taken over the randomness of* Auth, Hide *and* Sim*.*

If $\epsilon(\lambda) = \mathsf{negl}(\lambda)$*, we call the multi-key homomorphic authenticator scheme* statistically context hiding*. If* $\epsilon(\lambda) = 0$*, we call it* perfectly context hiding.

2.2.1 An Overview over Homomorphic Authenticator Schemes

The idea of linearly homomorphic authenticators was introduced in [55] and later refined in [73]. Freeman proposed stronger security definitions in [60].

Homomorphic authenticators can be split into two groups, privately verifiable homomorphic authenticators in the form of homomorphic MACs and publicly verifiable homomorphic authenticators in the form of homomorphic signatures.

Homomorphic MACs have been constructed for linear functions [3], quadratic functions [12] and arithmetic circuits [35, 104]. The latter constructions are built on primitives like fully homomorphic encryption or multilinear maps, for which efficient instantiations are still challenging.

In the case of homomorphic signatures, the first instantiation was provided by Boneh et.al. [24] based on the 2-3-Diffie Hellmann assumption. Later realizations are based on subgroup decision problems [7, 8], the k-Simultaneous Flexible Pairing Problem [9], the RSA problem [64] (offering only security against *weak adversaries*), the strong RSA problem [40], the Flexible DH Inversion problem [37], and the lattice based k-SIS problem [23].

The idea of homomorphic signatures with efficient verification was introduced in [41]. Intuitively, this means that the outcome of a computation can be checked faster by using the schemes verification algorithm than computing it oneself. However, this only holds in an *amortized* sense, as an expensive preprocessing phase has to be amortized over multiple datasets (see [12, 43]).

There are also homomorphic signatures beyond the linear case. Catalano et al. showed how to construct homomorphic signatures for arithmetic circuits of fixed depth from graded encoding schemes, a special type of multilinear maps [41]. Some lattice-based homomorphic signatures schemes [70], [59] support boolean circuits of fixed degree and therefore implicitly also arithmetic circuits of fixed degree. However, these schemes suffer the performance drawback of signing every single input bit.

2.3 Linearly Authenticated Encryption with Public Verifiability

Catalano et al. [43] introduced a cryptographic primitive which is called "**L**inearly Homomorphic **A**uthenticated **E**ncryption with **Pu**blic **V**erifiability" (LAEPuV). These schemes allow to evaluate a function over messages by evaluating a corresponding function over the encrypted messages such that the result can be verified by any third party. Below, we formally define LAEPuV schemes.

Definition 2.19. *(LAEPuV [43]). A LAEPuV scheme is a tuple of five PPT algorithms* $(\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *such that:*

$\mathsf{AKeyGen}(1^\lambda, n)$***:*** *It takes a security parameter* λ *and the maximum number* n *of encrypted messages in each dataset as input. It returns a key pair* $(\mathsf{sk}, \mathsf{pk})$*, where* sk *is the secret key for encrypting and signing and* pk *is the public key*

used for verification and evaluation. The message space $\mathcal{M}$, the ciphertext space $\mathcal{C}$, the input identifier space $\mathcal{T}$, and dataset identifier space $\mathcal{D}$ are implicitly defined by the public key pk.

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$***:*** *The input is a secret key* sk*, a dataset identifier $\Delta \in \mathcal{D}$, an input identifier $\tau \in \mathcal{T}$, and a message $m \in \mathcal{M}$. The output is a ciphertext c.*

$\mathsf{AEval}(\mathsf{pk}, f, \{c_i\}_{i=1}^n)$***:*** *The input is a public key* pk*, a linear function f, and a set of n ciphertexts $\{c_i\}_{i\in[n]} \in \mathcal{C}$. The output is a ciphertext c.*

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$***:*** *The input is a public key* pk*, a multi-labeled program $\mathcal{P}_\Delta$ containing a linear function f, and a ciphertext $c \in \mathcal{C}$. The output is either '1', i.e. the ciphertext is valid, or '0', i.e. the ciphertext is invalid.*

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$***:*** *It gets a secret key* sk*, a multi-labeled program $\mathcal{P}_\Delta$, and a ciphertext $c \in \mathcal{C}$ as input and outputs a message m if c is valid and $\perp$ if c is invalid, respectively.*

Intuitively, a LAEPuV scheme is correct if it satisfies three conditions. First, the decryption of a ciphertext yields the same message that was used to generate the ciphertext. Second, any ciphertext which is accepted by the verification algorithm can be decrypted, i.e. the decryption algorithm does not return $\perp$. Third, decrypting a ciphertext generated by the evaluation algorithm returns the message obtained by evaluating the function over the original messages. Below we formally define the correctness of LAEPuV schemes.

Definition 2.20 (Correctness [43])**.** *Let* $\mathsf{LAE} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *be a LAEPuV scheme. We say* LAE *is* correct *if the following three conditions all hold.*

1. *For any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n)$ honestly generated keypair, any message $m \in \mathcal{M}$, any dataset identifier $\Delta \in \mathcal{D}$, and any input identifier $\tau \in \mathcal{T}$ it holds with overwhelming probability*

$$\mathsf{ADecrypt}(\mathsf{sk}, \Delta, \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m), \mathcal{I}_\tau) = m$$

   where $\mathcal{I}_\tau$ is the labeled identity program (note that the identity is also linear).
2. *For any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n)$ and any ciphertext $c \in \mathcal{C}$ we have*

$$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1 \Leftrightarrow \exists m \in \mathcal{M} : \mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = m.$$

3. *Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, k)$ be a key pair, $\Delta \in \mathcal{D}$ be any dataset identifier, $m_1, \ldots, m_n \in \mathcal{M}$ be messages, and let $c_i \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau_i, m_i)$. For any admissible multi-labeled program $\mathcal{P}_\Delta = ((f_1, \ldots, f_n), \tau_1, \ldots, \tau_k, \Delta)$ it holds that*

$$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, \mathsf{AEval}(\mathsf{pk}, f, \{c_i\}_{i=1}^n)) = f(m_1, \ldots, m_n).$$

In the following we provide the definitions for the security of linearly homomorphic authenticated encryption with public verifiability (LAEPuV) schemes.

Security comes in two flavors. On the one hand privacy should be guaranteed, on the other hand no incorrect results should be accepted by such a scheme. Below we provide formalizations of these notions, as variants of indistinguishability under chosen ciphertext attacks and unforgeability under chosen ciphertext attacks, that are respectively fine tailored towards LAEPuV schemes.

These definitions are a slight modification of the original definitions by Catalano et al. [42] which directly use labeled programs in order to stay consistent with the rest of the Thesis. In the original labeled programs were only implicitly used.

Definition 2.21 (LH-IND-CCA [42] (adapted))**.** *Let* $\mathsf{LAE}=(\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ *be a LAEPuV scheme. We define the following experiment* $LH-IND-CCA_{\mathcal{H},\mathcal{A}}(1^\lambda, n)$ *between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$*:*

Setup: *The challenger runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n)$*. Then it initializes an empty list* L *and gives* pk *to the adversary* $\mathcal{A}$*.*

Queries I: $\mathcal{A}$ *can ask a polynomial number of both encryption and decryption queries. The former are of the form* (m, Δ, τ) *where* $m \in \mathcal{M}$ *is a message,* $\Delta \in \mathcal{D}$ *is a dataset identifier, and* $\tau \in \mathcal{T}$ *is an input identifier. The challenger computes* $c \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$*, gives* c *to* $\mathcal{A}$ *and updates the list* $L \leftarrow L \cup \{(m, \Delta, \tau)\}$*. If* L *already contains a query* $(\cdot, \Delta, \tau)$ *the challenger* $\mathcal{C}$ *will answer* $\perp$*.*
The latter queries are of the form $(\mathcal{P}_\Delta, c)$ *and* $\mathcal{A}$ *receives the output of* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$*. Note that this can be* $\perp$ *if* c *is not a valid ciphertext.*

Challenge: $\mathcal{A}$ *produces a challenge tuple* $(m_0, m_1, \Delta^*, \tau^*)$*. If a query of the form* $(\cdot, \Delta^*, \tau^*)$ *is contained in* L*, the challenger returns* $\perp$ *as before. The challenger chooses a random bit* $b \xleftarrow{\$} \{0,1\}$ *and gives*
$c^* \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta^*, \tau^*, m_b)$ *to* $\mathcal{A}$*. Then it updates the list* $L \leftarrow L \cup \{(m_b, \Delta^*, \tau^*)\}$*.*

Queries II: *This phase is carried out similar to the Queries I phase. Any decryption query* $(\mathcal{P}_{\Delta^*}, c)$ *with* $\mathcal{P}_{\Delta^*} = ((f_1, \ldots, f_n), \tau_1, \ldots, \tau_n, \Delta^*)$ *where* $f_{\tau^*} \neq 0$ *is answered with* $\perp$*. All other queries are answered as in phase Queries I.*

Output: *Finally* $\mathcal{A}$ *outputs a bit* $b' \in \{0,1\}$*. The challenger outputs '1'if* $b = b'$ *and '0'otherwise.*

We say that a LAEPuV scheme is LH-IND-CCA secure if for any PPT adversary $\mathcal{A}$ *we have*

$$|Pr[LH-IND-CCA_{\mathsf{LAE},\mathcal{A}}(1^\lambda, n) = 1] - 1/2| \leq \mathsf{negl}(\lambda).$$

Definition 2.22 (LH-Uf-CCA (adapted from[42]))**.** *A LAEPuV scheme is lienarly homomorphic unforgeable against chosen ciphertext attacks if the advantage of the PPT adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* λ*.*

Let LAE=*(*AKeyGen*,* AEncrypt*,* AEval*,* AVerify*,*ADecrypt*) be a LAEPuV scheme. We define the following experiment* $LH-Uf-CCA_{\mathcal{H},\mathcal{A}}(1^\lambda, n)$ *between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$*:*

Setup: *The challenger runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n)$*. Then it initializes an empty list* L *and gives* pk *to the adversary* $\mathcal{A}$*.*

Queries : $\mathcal{A}$ *can ask a polynomial number of both encryption and decryption queries. The former are of the form* (m, Δ, τ) *where* $m \in \mathcal{M}$ *is a message,* $\Delta \in \mathcal{D}$ *is a dataset identifier, and* $\tau \in \mathcal{T}$ *is an input identifier. The challenger computes* $c \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$*, gives* c *to* $\mathcal{A}$ *and updates the list* $L \leftarrow L \cup \{(\Delta, \tau, m, c, \mathcal{I}_\tau)$*. If* L *already contains a query* $(\Delta, \tau, \cdot, \cdot, \cdot)$ *the challenger* $\mathcal{C}$ *will answer* $\perp$*.*

The latter queries are of the form $(\mathcal{P}_\Delta, c)$ *and* $\mathcal{A}$ *receives the output of* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$*. Note that this can be* $\perp$ *if* c *is not a valid ciphertext.*

Forgery: *Finally* $\mathcal{A}$ *outputs* $(c^*, \mathcal{P}^*_{\Delta^*})$*.*

Let $L_{\Delta^*} = \{(\Delta^*, \tau, m, c, f)\} \subset L$ *be the set of entries in* L *where* $\Delta = \Delta^*$*. The adversary wins the game if* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) \neq \perp$ *and one of the following conditions holds:*

L_{Δ^*} *is empty*

f^* *(interpreted as a vector) is in the span of* $\{f | (\Delta^*, \cdot, \cdot, \cdot, f) \in L_{\Delta^*}\}$ *but for any* $\alpha_1, \ldots, \alpha_s$ *such that* $f^* = \sum_{i=1}^s \alpha_i f_i$ *it holds* $m^* \neq \sum_{i=1}^n \alpha_i m_i$

f^* *(interpreted as a vector) is not in the span of* $\{f | (\Delta^*, \cdot, \cdot, \cdot, f) \in L_{\Delta^*}\}$

Finally we define the advantage $\mathsf{Adv}^{LH-Uf_CCA}(\mathcal{A})$ *as the probability that* $\mathcal{A}$ *wins the game.*

There are known encryption schemes like Paillier [84] where the ciphertext space is larger then the message space. More precisely, the Paillier encryption scheme has message space $\mathbb{Z}_N$, where $N = pq$ for two primes p, q of equal size, while the corresponding ciphertext space is $\mathbb{Z}_{N^2}$.

Thus, a straightforward combination of the Paillier encryption scheme with a homomorphic signature scheme, requires a homomorphic signature scheme with message space $\mathbb{Z}_{N^2}$ which significantly decreases the performance of the scheme.

Catalano et al. [43] proposed a novel method which combines the Paillier encryption scheme and an arbitrary linearly homomorphic signature scheme that supports the same message space $\mathbb{Z}_N$. The high level idea is as follows. The message m is encrypted yielding a larger ciphertext C. This ciphertext is masked by multiplying it with a value R and the masked ciphertext is decrypted which gives a message a of the same size as the original message. Finally, a signature σ of the message a is generated using the homomorphic signature scheme. Functions are evaluated over the ciphertexts C, the messages a, and the signatures σ. To verify a computation, the scheme checks if σ is a valid signature of a and if the encryption of a still yields

the masked ciphertext of C. If both checks are true, the message m is obtained by decrypting the ciphertext C.

Below we describe the LAEPuV framework by Catalano et al. [43] which combines the Paillier encryption scheme with an arbitrary homomorphic signature scheme $\mathsf{HS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$.

Construction 2.23.

$\mathsf{AKeyGen}(1^\lambda, n)$*: On input a security parameter λ and an integer n, the algorithm chooses two (safe) primes of size $\lambda/2$, sets $N \leftarrow pq$, and chooses an element $g \in \mathbb{Z}^*_{N^2}$ of order N. It chooses a linearly homomorphic signature scheme $\mathsf{HS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ with message space N. It runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ as well as $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Note that we implicitly give the message space as an additional argument to the key generation of the homomorphic signature scheme as it not necessarily use the factorization of N as its secret key. It chooses a hash function $H \leftarrow \mathcal{H}$, and returns the key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = (p, q, \mathsf{sk}')$ and $\mathsf{pk} = (N, H, g, \mathsf{ek}, \mathsf{vk})$.*

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$*: On input a secret key sk, a dataset identifier Δ, an input identifier $\tau \in \mathcal{T}$, and a message $m \in \mathbb{Z}_N$, the algorithm chooses $\beta \xleftarrow{\$} \mathbb{Z}^*_{N^2}$, computes $C \leftarrow g^m \beta^N \mod N^2$, sets $R \leftarrow H(\Delta||\tau)$, and computes $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$ such that $g^a b^N = CR \mod N^2$ by invoking the following steps [84]:*

- *Obtain a by decrypting CR using the Paillier cryptosystem [84].*
- *Compute $c_* \leftarrow CRg^{-a} \mod N$.*
- *Set $b \leftarrow c_*^{N^{-1} \bmod L} \mod N$, where $L = \mathsf{lcm}(p-1, q-1)$.*

Finally, the algorithm computes the signature $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}', \Delta, \tau, a)$ of a and returns the ciphertext $c = (C, a, b, \sigma)$.

$\mathsf{AEval}(\mathsf{pk}, \Delta, f, \{c_i\}_{i\in[n]})$*: On input a public key pk, a dataset identifier Δ, a linear function f, and n ciphertexts $\{c_i\}_{i\in[n]}$, with $c_i = (C_i, a_i, b_i, \sigma_i)$, the algorithm computes*

$$C \leftarrow \prod_{i=1}^{n} C_i^{f_i} \mod N^2$$

$$a \leftarrow \sum_{i=1}^{n} f_i a_i \mod N$$

$$b \leftarrow \prod_{i=1}^{n} b_i^{f_i} \mod N^2$$

$$\sigma \leftarrow \mathsf{Eval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$$

and returns the ciphertext $c = (C, a, b, \sigma)$.

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$*: On input a public key* pk*, a multi-labeled program* $\mathcal{P}_\Delta$ *containing a linear function* f*, and a ciphertext* $c = (C, a, b, \sigma)$*, algorithm checks if*

$$\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma) = 1$$
$$g^a b^N = C \prod_{i=1}^{n} H(\Delta||\tau_i)^{f_i} \mod N^2$$

If both equations are satisfied, the algorithm returns 1*, i.e. the ciphertext is valid, otherwise, it returns* 0*, i.e. the ciphertext is invalid.*

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$*: On input a secret key* sk*, a multi-labeled program* $\mathcal{P}_\Delta$*, and a ciphertext* $c = (C, a, b, \sigma)$*, the algorithm returns* $\perp$ *if* $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = 0$*. Otherwise, it returns* m *obtained by decrypting* C *using the Paillier cryptosystem.*

We will present the two theorems arguing about the security of this construction.

Theorem 2.24 ([43, Theorem 6])**.** *In the random oracle model, if the* DCR *assumption (see Definition 2.44) holds,* H *is a random oracle, and* $\mathsf{HS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *is a linearly homomorphic signature scheme over* $\mathbb{Z}_N$ *that is unforgeable in the sense of Def. 2.11, Construction 2.23 is LH-IND-CCA secure according to Definition 2.21.*

Theorem 2.25 ([43, Theorem 7])**.** *If* $\mathsf{HS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *is a linearly homomorphic signature scheme over* $\mathbb{Z}_N$ *that is unforgeable in the sense of Def. 2.11, then Construction 2.23 is LH-Uf-CCA secure according to Definition 2.22.*

2.4 Secret Sharing

Shamir secret sharing allows a *data owner* to distribute a message among a set of *shareholders*, such that the message can only be reconstructed if a qualified subset of these shareholders collaborates. At the same time no other subset can learn any information about the message distributed. Let N be the total number of shareholders, $i \in [N]$ be the unique ID of shareholder i, $t \leq N$ be the threshold required for reconstruction, and $\mathbb{F}_p$ be a field with $p > N$ elements. Then, Shamir secret sharing can be defined via the following two algorithms.

Definition 2.26 (Shamir Secret Sharing [97])**.** *Let* N *be the total number of shareholders,* $i \in [N]$ *be the unique ID of shareholder* i*,* $t \leq N$ *be the threshold required for reconstruction, and* $\mathbb{F}_p$ *be a field with* $p > N$ *elements. Then Shamir secret sharing consists of the following algorithms.*

$\mathsf{SShare}(m)$: *Given a message $m \in \mathbb{F}_p$ as input the algorithm chooses a polynomial $f(x) = m + a_1 + a_2 x + \cdots + a_{t-1}x^{t-1}$, where the message m is the free coefficient and the other coefficients $a_1, \cdots, a_{t-1}$ are chosen uniformly at random. Then it computes N shares s_i, for $i \in [N]$, where $s_i = f(j)$ is sent to shareholder i.*

$\mathsf{SReconstruct}(\mathcal{B}, \{s_i\}_{i\in\mathcal{B}})$: *It takes as input a subset $\mathcal{B} \subset \{1, \ldots, N\}$ of shareholders and corresponding shares s_i, $\forall i \in \mathcal{B}$. If $|\mathcal{B}| < t$ it outputs $\perp$. Otherwise it reconstructs the unique interpolation polynomial $f^*(x)$ of degree $t-1$ in $\mathbb{F}_p[x]$ and returns message $f^*(0) = m^*$.*

Note that there exists a reconstruction vector $(\{w_i\}_{i\in\mathcal{B}})$ such that $\sum_{i\in\mathcal{B}} w_i s_i = m$. Since for Shamir secret sharing the Lagrange Interpolation *formula is used to reconstruct the message m the reconstruction vector is defined as $w_i = \prod_{j\in\mathcal{B}, j\neq i} \frac{j}{j-i}$, for $i \in \mathcal{B}$. Furthermore, shares generated with polynomials of degree $t-1$ are called t-reconstructing shares.*

2.5 Commitment Schemes

Commitment schemes, particularly homomorphic commitment schemes, are a basic building block in cryptography.

A commitment scheme can be seen as the cryptographic analogue of a sealed envelope. It allows a party to fix any chosen message m without other parties learning about m. At a later point, the party can chose to reveal the message m and show that the revealed message is the one that was fixed earlier. We provide the formalizations used in this work.

Definition 2.27 (Commitment Scheme)**.** *A commitment scheme* Com *is a tuple of the following algorithms (*CSetup, Commit, Decommit*):*

$\mathsf{CSetup}(1^\lambda)$: *On input a security parameter λ, this algorithm outputs a commitment key* CK*. We implicitly assume that every algorithm uses this commitment key, leaving it out of the notation.*

$\mathsf{Commit}(m, r)$: *On input a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, it outputs the commitment C and the decommitment d.*

$\mathsf{Decommit}(m, d, C)$: *On input a message $m \in \mathcal{M}$, decommitment d, and a commitment C it outputs '1' or '0'.*

Definition 2.28. *Let $\mathcal{F}$ be a class of functions. A commitment scheme* $\mathsf{Com} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit})$ *is $\mathcal{F}$-homomorphic if there exists an algorithm* CEval *with the following properties:*

$\mathsf{CEval}(f, C_1, \ldots, C_n)$: *On input a function $f \in \mathcal{F}$ and a tuple of commitments C_i for $i \in [n]$, the algorithm outputs C^*.*

Correctness: *For every* $m_i \in \mathcal{M}$, $r_i \in \mathcal{R}$, $i \in [n]$ *with* $(C_i, d_i) \leftarrow \mathsf{Commit}(m_i, r_i)$ *and* $C^* \leftarrow \mathsf{CEval}(f, C_1, \ldots, C_n)$, *there exists a unique function* $\hat{f} \in \mathcal{F}$, *such that* $\mathsf{Decommit}(f(m_1, \ldots, m_n), \hat{f}(m_1, \ldots, m_n, d_1, \ldots d_n), C^*) = 1$.

2.5.1 An Overview over Commitment Schemes

Commitment schemes are a convenient tool to add verifiability to various processes, such as secret sharing [88], multi-party computation [16], or e-voting [83]. The most well-known and widely used commitment schemes used to provide verifiability are Pedersen's commitments [88] and Feldmann's commitments [57]. Libert et al. [79] introduce the notion of *functional commitments*. Functional commitments are commitments to a tuple of messages that can then be opened to any linear combination of these messages.

2.6 Verifiable Computing

Definition 2.29 (Verifiable Computing Scheme [62])**.** *A* Verifiable Computing Scheme VC *is a tuple of the following probabilistic polynomial-time (PPT) algorithms:*

$\mathsf{VKeyGen}(1^\lambda, f)$: *The probabilistic key generation algorithm takes a security parameter* λ *and the description of a function* f. *It generates a secret key* sk, *a corresponding verification key* vk, *and a public evaluation key* ek *(that encodes the target function* f*) and returns all these keys.*

$\mathsf{ProbGen}(\mathsf{sk}, x)$: *The problem generation algorithm takes a secret key* sk *and data* x. *It outputs a public value* σ_x *which encodes the data* x *and a corresponding decoding value* ρ_x.

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *The computation algorithm takes the evaluation key* ek *and the encoded input* σ_x. *It outputs an encoded version* σ_y *of the function's output* $y = f(x)$.

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: *The verification algorithm obtains a verification key* vk *and the decoding value* ρ_x. *It converts the encoded output* σ_y *into the output of the function* y. *If* $y = f(x)$ *holds, it returns* y *or outputs* $\perp$ *indicating that* σ_y *does not represent a valid output of* f *on* x.

Definition 2.30 (Correctness [62])**.** *A verifiable computing scheme* VC *is correct if for any choice of* f *and output* $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{VKeyGen}(1^\lambda, f)$ *of the key generation algorithm it holds that* $\forall\ x \in \mathsf{Domain}(f)$, *if* $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$ *and* $y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$, *then* $y = f(x) \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$.

In the original work on non-interactive verifiable computing Gennaro et al. [62] only considered privately verifiable computing schemes as defined below.

Definition 2.31 (Privately Verifiable Computing Scheme)**.** *If* $\mathsf{sk} = \mathsf{vk}$ *and C needs to keep* ρ_x *private,* VC *is called a* privately verifiable computing scheme*.*

Clearly, such a scheme requires the client to run the verification algorithm. Later in [87], Parno et al. introduced the notion of publicly verifiable computing schemes.

Definition 2.32 (Publicly Verifiable Computing Scheme)**.** *If* $\mathsf{sk} \neq \mathsf{vk}$*,* VC *is called a* publicly verifiable computing scheme*.*

It allows to hand out vk to third parties without revealing sk. Therefore, everyone with knowledge of vk and ρ_x can verify the correctness of the server's computation.

Intuitively the difference between the two notions is that in privately verifiable computing the client keeps its verification key secret. It follows that only the client can act as verifier. Note that in privately verifiable computing schemes revealing the verification key often leads to a loss of security. More precisely, knowledge of the verification key allows the server computing a wrong result leading to a correct verification proof. In publicly verifiable computing, on the other hand, knowledge of the verification key does not help a malicious server to forge an incorrect result. Thus, it can be published allowing not only the client but anyone to act as verifier and to check the correctness of a performed computation.

2.7 Properties of Verifiable Computing Schemes

In this section a definition for security and efficiency is given. We will mainly follow the approach of Gennaro et al. [62], who were the first to define verifiable computing schemes. In addition, we also integrate some later proposals to obtain stronger security definitions, e.g. adaptive security presented in [20].

2.7.1 Security

Intuitively a verifiable computing scheme VC is secure, if a malicious server cannot persuade the verification algorithm to output $y^* \neq f(x)$ except with negligible probability. Formally, we define the following two experiments. We distinguish between two types of adversaries, a weak adversary and an adaptive adversary. The weak adversary [62] only has oracle access to $\mathsf{ProbGen}$ but is not allowed to call Verify in the privately verifiable computing setting. It can only try once to have an incorrect result verified as correct but must never learn the client's acceptance bit, since this information might be used to produce subsequent forgeries.

An adaptive adversary [20] can run $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}$ multiple times, and learn about the client's acceptance bit and adapt its forgeries accordingly.

In the non-adaptive case the adversaries $\mathcal{A}$'s advantage is defined as

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}[\mathsf{VC}, f, \lambda]$:
  $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, f)$
  for $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$ **do**
    $x_i \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots x_{i-1}, \sigma_1, \ldots, \sigma_{i-1})$
    $(\sigma_i, \rho_i) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$
  end for
  $(i, \sigma_y^*) \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots, x_\ell, \sigma_1, \ldots, \sigma_\ell)$
  $y^* \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_i, \sigma_y^*)$
  if $y^* \neq \perp \ \wedge \ y^* \neq f(x)$ **then**
    return 1
  else
    return 0
  end if

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Verify}}(\mathsf{VC}, f, \lambda) = \Pr\Big[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}[\mathsf{VC}, f, \lambda] = 1\Big].$$

So in practice this type of adversary is acceptable if a client aborts the protocol once it detects an incorrect result.

An adaptive adversaries $\mathcal{A}$'s advantage is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AdaptVerify}}(\mathsf{VC}, f, \lambda) = \Pr\Big[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda] = 1\Big].$$

From this the security definition for verifiable computing schemes follows.

Definition 2.33 (Security [20, 62])**.** *A verifiable computing scheme* VC *is (weakly) secure if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Verify}}(\mathsf{VC}, f, \lambda) \leq \mathsf{negl}(\lambda)$$

and adaptively secure if

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AdaptVerify}}(\mathsf{VC}, f, \lambda) \leq \mathsf{negl}(\lambda).$$

2.7.2 Efficiency

Finally we are interested in using verifiable computing schemes by means of delegating computations. For this we want the work performed by the client and the verifier to be less than computing the function on their own.

Definition 2.34 (Efficiency [62])**.** *A verifiable computing scheme provides* efficiency *if for any* x *and any* σ_y*, the time required for* $\mathsf{VKeyGen}(1^\lambda, f)$ *plus the time required for* $\mathsf{ProbGen}(\mathsf{sk}, x)$ *plus the time required for* $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$ *is* $o(T)$*, where* T *is the time required to compute* $f(x)$*.*

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda]$:
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{VKeyGen}(f, 1^{\lambda})$
for $j = 1, \ldots, m = \mathsf{poly}(\lambda)$ **do**
  for $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$ **do**
    $x_i \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots x_{i-1}, \sigma_1, \ldots, \sigma_{i-1}, \delta_1, \ldots, \delta_{j-1})$
    $(\sigma_i, \rho_i) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$
  end for
  $(i, \sigma_y^*) \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots, x_\ell, \sigma_1, \ldots, \sigma_\ell, \delta_1, \ldots, \delta_{j-1})$
  $y^* \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_i, \sigma_y^*)$
  if $y^* \neq \perp \ \wedge \ y^* \neq f(x)$ **then**
    $\delta_j := 1$
  else
    $\delta_j := 0$
  end if
end for
if $\exists\ j$ such that $\delta_j = 1$ **then**
  return 1
else
  return 0
end if

A slightly relaxed definition is the following.

Definition 2.35 (Amortized Efficiency)**.** *A verifiable computing scheme provides* amortized efficiency *if it permits efficient verification. This implies that for any x and any σ_y, the time required for $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.*

Note that in literature amortized efficiency has been defined ambiguously. We use here a broad version that ensures that the minimal requirements for outsourceability are met.

Intuitively the difference between efficiency and amortized efficiency is the cost of the preprocessing phase. Efficient verifiable computing schemes allow a verifier to verify the correctness of a computation more efficiently than performing the computation by itself, including the preprocessing phase performed by the client. Some verifiable computing schemes have an expensive preprocessing phase, but still provide an efficient verification phase. Since the preprocessing phase only has to be performed once and might not be time critical in many applications, we classify them as verifiable computing schemes providing amortized efficiency.

One aspect that also impacts the practicality of all verifiable computing schemes is the server's overhead to evaluate a computation using $\mathsf{Compute}$ versus natively

executing it. Note that this does not affect the computation complexity for the client or the verifier. So far in literature this aspect has not been rigorously covered and is therefore not considered in our efficiency analysis.

Existing verifiable computing schemes are compared in detail in Section 3.2.

2.8 Cryptographic Assumptions

We recall the computational assumptions on which our schemes are based.

Definition 2.36 (DL)**.** *Let $\mathcal{G}$ be a generator of cyclic groups of order p and let $\mathbb{G} \xleftarrow{\$} \mathcal{G}(1^\lambda)$. We say the* Discrete Logarithm *assumption (DL) holds in $\mathbb{G}$ if there exists no PPT adversary $\mathcal{A}$ that, given (g, g^a) for a random generator $g \in \mathbb{G}$ and random $a \in \mathbb{Z}_p$, can output a with more than negligible probability, i.e. if* $\Pr\left[a \leftarrow \mathcal{A}(g, g^a) \mid g \xleftarrow{\$} \mathbb{G},\ a \xleftarrow{\$} \mathbb{Z}_p\right] = \mathsf{negl}(\lambda)$.

Definition 2.37 (Asymmetric bilinear groups)**.** *An* asymmetric bilinear group *is a tuple* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$*, such that:*

- $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *are cyclic groups of prime order* p*,*
- $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *are generators for their respective groups,*
- *the DL assumption holds in* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$*,*
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is bilinear, i.e.* $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab} \ \ \forall\ a, b \in \mathbb{Z}$*,*
- e *is non-degenerate, i.e.* $e(g_1, g_2) \neq 1$*, and*
- e *is efficiently computable.*

We will write $g_t = e(g_1, g_2)$*.*

In case of symmetric pairings where $\mathbb{G}_1 \simeq \mathbb{G}_2$ *we write* $\mathbb{G}$ *for both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *for the sake of convenience.*

Definition 2.38 (DDH)**.** *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the* Decisional Diffie–Hellman *assumption (DDH) holds in $\mathbb{G}_1$ if, for every PPT adversary $\mathcal{A}$,*

$$\left|\Pr\left[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_1^{xy}) \mid x, y \xleftarrow{\$} \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_1^z) \mid x, y, z \xleftarrow{\$} \mathbb{Z}_p\right]\right| = \mathsf{negl}(\lambda).$$

Definition 2.39 ($co-DHP^*$ [44])**.** *Let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ *be a description of a bilinear group. We say the Computational Diffie-Hellman assumption holds in* bgp*, if there exists no ppt adversary $\mathcal{A}$ that given* $(\mathsf{bgp}, g_1^a, g_1^b, g_2^b)$ *where* $a, b \xleftarrow{\$} \mathbb{Z}_p$ *can output* g_1^{ab} *with more than negligible probability.*

Definition 2.40 (Double Pairing Assumption in $\mathbb{G}_1$ (DBP_1 [1]))**.** *Let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ *and* $R, Z \xleftarrow{\$} \mathbb{G}_1$*. Any PPT adversary* $\mathcal{A}$ *can produce* $(G_R, G_Z) \in \mathbb{G}_2^2 \backslash \{(1,1)\}$ *such that* $1 = e(R, G_R) \cdot e(G, G_Z)$ *only with a probability negligible in* λ*.*

We have an analogous computational problem in $\mathbb{G}_2$.

Definition 2.41 (Double Pairing Assumption in $\mathbb{G}_2$ (DBP_2 [1]))**.** *Let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ *and* $R, Z \xleftarrow{\$} \mathbb{G}_2$*. Any PPT adversary* $\mathcal{A}$ *can produce* $(G_R, G_Z) \in \mathbb{G}_1^2 \backslash \{(1,1)\}$ *such that* $1 = e(G_R, R) \cdot e(G_Z, Z)$ *only with a probability negligible in* λ*.*

It is known that DBP_1 implies the Decisional Diffie–Hellman assumption in $\mathbb{G}_1$ and DBP_2 implies the Decisional Diffie–Hellman assumption in $\mathbb{G}_2$, and the security reductions are tight [2].

We also use the Flexible Diffie–Hellman Inversion hardness assumption, introduced by Catalano, Fiore and Nizzardo [37]. In the extended version of their CRYPTO2015 paper, they formally investigate the hardness of this assumption and analyse it in the generic group model.

Definition 2.42 (FDHI [37])**.** *Let* $\mathcal{G}$ *be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the* Flexible Diffie–Hellman Inversion *(FDHI) assumption holds in* bgp *if, for every PPT adversary* $\mathcal{A}$*,*

$$\Pr\left[W \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\} \wedge W' = W^{\frac{1}{z}} \mid (W, W') \leftarrow \mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}) \mid z, r, v \xleftarrow{\$} \mathbb{Z}_p\right] = \mathsf{negl}(\lambda).$$

Definition 2.43 (Factorization Assumption)**.** *Let* $N = pq$ *be a random RSA modulus of length* λ*, where* $\lambda \in \mathbb{N}$*. We say that the factorization assumption holds if for any PPT adversary* $\mathcal{A}$ *we have* $\Pr[(p, q) \leftarrow \mathcal{A}(N)] = \mathsf{negl}(\lambda)$

Definition 2.44 (DCRA [84])**.** *Let* N *be the product of two (safe) primes, i.e.* $N = pq$*. We say the Decisional composite residuosity assumption (DCRA) holds if there exists no ppt adversary* $\mathcal{A}$ *that can distinguish between an element drawn uniformly random from the set* $\mathbb{Z}_{N^2}^*$ *and an element from the set* $\{z^N | z \in \mathbb{Z}_{N^2}^*\}$*, that is the set of the* N*-th residues modulo* N^2*.*

Definition 2.45 (Strong RSA [14])**.** *Let* $N = pq$ *be a random RSA modulus of length* λ*, where* $\lambda \in \mathbb{N}$*, and* $z \in \mathbb{Z}_N$ *a random element in* $\mathbb{Z}_N$*. We say that the Strong RSA assumption holds if for any PPT adversary* $\mathcal{A}$ *we have* $\Pr[(y, e) \leftarrow \mathcal{A}(N, z) : y^e = z \mod N \wedge e \neq 1] = \mathsf{negl}(\lambda)$.

3 Classifying Verifiable Computing Schemes by Their Privacy Properties

Today, it is common practice to outsource large-scale computations to the cloud. In such a situation, it is desirable to be able to verify the outsourced computation. The background for the field of verifiable computing was discussed in Section 2.6. In addition to the basic properties discussed there, further properties are required when dealing with sensitive data. Particularly sensitive data, e.g. medical data, must remain private even in the long term. Thus, privacy that depends on the computational hardness of certain problems is insufficient. Here everlasting, i.e. information-theoretic privacy is required.

To account for the different types of privacy threats arising in such scenarios we distinguish between four different privacy notions. We first distinguish by *which information* is private - input or the outcome of the computation. We then distinguish by *the attacker* we want to prevent from learning sensitive information. In the case of verifiable computing, this can either be the server performing the computation or the third-party verifier verifying the correctness.

Organization. In this chapter we first provide definitions for multiple privacy notions related to verifiable computing in Section 3.1. We then give an overview of existing verifiable computing schemes and their properties - in particular their privacy properties in Section 3.2.

Publications. This section is based on publication [S1].

3.1 Privacy

Verifiable computing can guarantee the integrity of a computation. Beyond that, a desirable property is to protect the secrecy of the client's inputs towards the server(s)

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t]$
$(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(f, 1^\lambda)$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek})$, s.t. $f(x_0) = f(x_1)$
$(\sigma_0, \rho_0) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_1, \rho_1) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$b \xleftarrow{\$} \{0, 1\}$
$\sigma_{y_b} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_b})$
$\mathcal{B} \leftarrow \mathcal{A}$ with $\mathcal{B} \subset \{1, \ldots, n\}$ and $|\mathcal{B}| = t - 1$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{ek}, x_0, x_1, \{\sigma_{y_b,j}\}_{j\in\mathcal{B}})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

and when using a publicly verifiable scheme also towards the verifiers. To formally define *input privacy w.r.t the servers* we define experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Server}}}$, during which the adversary controls t servers. We use the oracle $\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},x)}$ which calls $\mathsf{ProbGen}(\mathsf{sk}, x)$ to obtain (σ_x, ρ_x) and only returns the public part σ_x.

In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1 and is given the encoding of one of the two inputs chosen at random. The adversary then must determine which input has been encoded. During this process, the adversary may request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as
$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = \left|\Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t] = 1\right] - \frac{1}{2}\right|$.

Definition 3.1 (Input privacy w.r.t. the server [62])**.** *A verifiable computing scheme $\mathcal{VC}$ provides unconditional input privacy with respect to the server if any computationally unbounded adversary $\mathcal{A}$ has $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = 0$.*

If the advantage is negligible, we say the scheme provides *computational input privacy with respect to the server.*

We give an analogous definition for output privacy. To formally define *output privacy w.r.t the server*, we define experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}$, during which the adversary controls t servers. We use the oracle $\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},x)}$ which calls $\mathsf{ProbGen}(\mathsf{sk}, x)$ to obtain (σ_x, ρ_x) and only returns the public part σ_x.

In this experiment, the adversary first receives the public evaluation key for the scheme. Then, it selects two inputs x_0, x_1. It is given the results of the computation $y_0 = f(x_0), y_1 = f(x_1)$ and is given the encoding of one of the two inputs chosen at random. The adversary corrupts t servers and receives their share of the encoding

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t]$
$(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(f, 1^{\lambda})$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{ek})$
$(\sigma_{x_0}, \rho_{x_0}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_{x_1}, \rho_{x_1}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$b \xleftarrow{\$} \{0, 1\}$
$\sigma_{y_b} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_b})$
$\mathcal{B} \leftarrow \mathcal{A}$ with $\mathcal{B} \subset \{1, \ldots, n\}$ and $|\mathcal{B}| = t - 1$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{ek}, y_0, y_1, \{\sigma_{y_b,j}\}_{j \in \mathcal{B}})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

and then must determine which encoded output has been computed. During this process, the adversary may request the encoding of any input of its choice.

We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}[\mathcal{VC}, f, \lambda, t] = 1\right] - \frac{1}{2} \right|$$

Definition 3.2 (Output privacy w.r.t. the server)**.** *A verifiable computing scheme $\mathcal{VC}$ provides unconditional output privacy with respect to the server if any computationally unbounded adversary $\mathcal{A}$ has $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy_{Server}}}(\mathcal{VC}, f, \lambda, t) = 0$.*

If the advantage is merely negligible, we say the scheme provides *computational output privacy with respect to the server.*

If we have a publicly verifiable computing scheme a third-party verifier might try to learn about the input data from the publicly available verification data. To formally define *input privacy w.r.t a third-party verifier* we define the following experiment.

In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1 and is given the encoding of one of the two inputs chosen at random. The adversary then must determine which input has been encoded. Note that during this process the adversary is allowed to request the encoding of any input of its choice.

We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy_{Verifier}}}(\mathsf{VC}, f, \lambda) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy_{Verifier}}}[\mathsf{VC}, f, \lambda] = 1\right] - \frac{1}{2} \right|.$$

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}[\mathsf{VC}, f, \lambda]$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{VKeyGen}(f, 1^{\lambda})$
$(x_0, x_1) \leftarrow \mathcal{A}^{O^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{vk})$
$(\sigma_{x_0}, \rho_{x_0}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_{x_1}, \rho_{x_1}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$b \xleftarrow{\$} \{0, 1\}$
$\sigma_{y_b} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_b})$
$b^* \leftarrow \mathcal{A}^{O^{\mathsf{ProbGen}(\mathsf{sk},\cdot)}}(\mathsf{vk}, x_0, x_1, \sigma_{y_b}, \rho_{y_b})$
if $b^* = b$ **then**
  return 1
else
  return 0
end if

Definition 3.3 (Input Privacy w.r.t. the Verifier)**.** *A verifiable computing scheme* VC *provides input privacy if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{In-Privacy}_{\mathsf{Verifier}}}(\mathsf{VC}, f, \lambda) \leq \mathsf{negl}(\lambda).$$

If the advantage is negligible, we say the scheme provides *computational input privacy with respect to the verifier.*

Note that for authenticator-based verifiable computing schemes [12] this captures all the variations of the context hiding property (see Definitions 5.1 to 2.18) discussed in Section 2.2.

For a publicly verifiable computing scheme, it is interesting to show correctness of a computation without revealing its outcome. A third-party verifier might try to learn about the output data from the publicly available verification data. To formally define *output privacy w.r.t a third-party verifier*, we define the following:

Definition 3.4 (Output privacy w.r.t. the verifier)**.** *A verifiable computing scheme* $\mathcal{VC}$ *provides unconditional output privacy with respect to the verifier if there exist additional algorithms* (HideCompute, HideVerify, Decode)*:*

- $\mathsf{HideCompute}(\mathsf{ek}, \sigma_x)$: *The computation algorithm takes the evaluation key* ek *and the encoded input* σ_x*. It outputs an encoded version* $\tilde{\sigma}_y$ *of the function's output* $y = f(x)$*.*
- $\mathsf{HideVerify}(\mathsf{vk}, \tilde{\sigma_y})$: *The verification algorithm obtains a verification key* vk *and an encoding* $\tilde{\sigma}_y$*. It outputs either* $\perp$*, or* $\hat{\sigma}_y$*.*
- $\mathsf{Decode}(\mathsf{vk}, \rho_x, \hat{\sigma}_y, \sigma_y)$: *It takes as input the verification key* vk*, a decoding value* ρ_x*, and encoded values* $\hat{\sigma}_y, \sigma_y$*. It outputs either* $\perp$ *indicating that* σ_y *does not represent a valid output of* f *on* x *or it returns* y*.*

Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{out-Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda]$
$(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(f, 1^\lambda)$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek})$
$(\sigma_{x_0}, \rho_{x_0}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
$(\sigma_{x_1}, \rho_{x_1}) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
$\tilde{\sigma_{y_0}} \leftarrow \mathsf{HideCompute}(\mathsf{ek}, \sigma_{x_0})$
$\tilde{\sigma_{y_1}} \leftarrow \mathsf{HideCompute}(\mathsf{ek}, \sigma_{x_1})$
$b \xleftarrow{\$} \{0, 1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{vk},\cdot)}}(\mathsf{ek}, y_0, y_1, \tilde{\sigma_{y_b}}, \rho_{x_b})$
if $b^* = b$ **then**
return 1
else
return 0
end if

and the following two properties hold:

Correctness *For any security parameter λ, and any admissible function f, let $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, f)$, and any input x, we have for $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$, $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$,*
and
$\tilde{\sigma}_y \leftarrow \mathsf{HideCompute}(\mathsf{ek}, \sigma_x)$, $\hat{\sigma}_y \leftarrow \mathsf{HideVerify}(\mathsf{vk}, \tilde{\sigma}_y)$
$\Pr[\mathsf{Decode}(\mathsf{vk}, \rho_x, \hat{\sigma_y}, \sigma_y) \neq \mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)] = \mathsf{negl}(\lambda)$.

Privacy *We describe the experiment $\boldsymbol{EXP}_{\mathcal{A}}^{\mathsf{out-Privacy}_{\mathsf{Verifier}}}$.*
In this experiment, the adversary first receives the public verification key for the scheme. Then, it selects two inputs x_0, x_1. It is given the results $y_0 = f(x_0), y_1 = f(x_1)$ and is given the encoding of one of the two outputs chosen at random. The adversary then must determine which input has been encoded. During this process, the adversary may request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = \left| \Pr\left[\boldsymbol{EXP}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}[\mathcal{VC}, f, \lambda] = 1 \right] - \frac{1}{2} \right|.$$

for any computationally unbounded adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Out-Privacy}_{\mathsf{Verifier}}}(\mathcal{VC}, f, \lambda) = 0$.

If the advantage is negligible, we say the scheme provides *computational output privacy with respect to the verifier.*

3.2 An Overview over Verifiable Computing Schemes

In this section, we summarize verifiable computing scheme and compare them, in particular with respect to their privacy properties. The first property examined is which *function class* the scheme supports. Some support (subsets of) arithmetic circuits, while others can also deal with stateful operations or general loops, i.e. without needing to know the length of the loop during the preprocessing stage. Furthermore, we specify which type of *adversary* the solution can cope with. Some schemes are secure against a strong adversary (S), some are only secure against a weak adversary (W), and for some approaches the security level has not been analyzed yet ($\emptyset$). Note that this corresponds to the adaptive and weak adversaries described in Def. 2.33.

In addition, we show which *primitives* the construction relies on, since most of them come with further assumptions regarding security. Furthermore, in some scenarios it might be preferable that the scheme provides a certain level of *privacy*. Depending on the type of data, a scheme may either ensure input privacy (I), output privacy (O), input-output privacy (I/O), or no privacy at all ($\times$) with regard to the server. Several solutions are tailored to *private verification*, i.e. where the verification can only be performed by the data owner. However, in some scenarios the verification must be performed by a party different from the owner, requiring the scheme to be *publicly verifiable*. For computing schemes that are publicly verifiable we also highlight whether they provide privacy towards the verifier, i.e. the public. Thus, for a publicly verifiable computing scheme we further distinguish whether it provides input privacy (I), output privacy (O), input-output privacy (I/O), or no privacy at all ($\times$) with respect to the verifier. For privately verifiable computing schemes this notion is obviously not applicable (NA).

To be successfully applied in practice, a scheme also needs to provide *efficiency*. We define a verifiable computing scheme as efficient (E), if the time required for preprocessing and verification is $o(T)$, where T is the time required to compute the function. If only the verification can be performed in $o(T)$, then the computing scheme only provides amortized efficiency (A). Sometimes it is not possible to make a general statement about a scheme's attributes as they are dependent on choices of primitives (D). The abbreviations introduced here are summarized in Table 3.1.

In the setting of proof based verifiable computing a *(super-)polynomial-time prover* wants to convince a *computationally bounded verifier* of the validity of some statement in an **NP**-language. In the context of verifiable computing the prover is the server performing a given computation and the statement represents the correctness of the computed result. To achieve this goal, rather theoretical tools,

Category	Abbreviation	Explanation
Adversary	S	Strong adversary
	W	Weak adversary
Privacy	I	Input privacy
	O	Output Privacy
	I/O	Input-output privacy
	inf.	information-theoretic
	×	no privacy
	NA	no information available
Efficiency	E	Efficient
	A	Amortized efficient
General	D	Dependent on primitives

Table 3.1: Used abbreviations

such as *interactive proof systems* [68] and *probabilistically checkable proofs (PCP)* [6, 10] were used. While the application to verifiable computing scenarios have already been mentioned in very early works, the solutions from these theoretical tools were not suitable for any practical application. Later work relaxed these potentially super-polynomial provers to polynomially bounded provers to obtain (typically more efficient) *argument systems* [30]. Furthermore, this line of work has been improved by both theoretical refinements and suitable implementations.

As shown by Table 3.2, the only proof-based approach that provides an efficient generation and verification process is the one proposed in [101]. This scheme, however, only supports a very restricted class of circuits. The other PCP or linear PCP based constructions support larger classes of programs, but achieve only amortized efficiency. In addition, all these approaches are interactive, i.e. require multi-round interaction between the server and the client. To reduce the server's overhead later solutions are non-interactive. Backes et al.'s proposal [11] even achieves input privacy towards the verifier and provides public verifiability. However, all non-interactive proof based schemes use so called QAPs [63] and are based on *non-falsifiable* assumptions of knowledge, i.e. assumptions that cannot be efficiently denied. As shown in [67] it is actually impossible to build a SNARG (e.g. using QAPs) that is based solely on falsifiable assumptions. This raises some questions on the security of these schemes. In fact, although it has been shown that PCPs and QAPs are secure against an adaptive adversary, it has not been proven that the same holds true for the verifiable computing scheme using this primitive. For some of the schemes offering public verifiability two variations are described – one with and one without input privacy. In this case we only denote the version *with input privacy*.

Scheme	Function Class	PrS	E	VER	PrV
[101]/[100]	Circuits of polylog. depth	×	E	✓	NA
[102]	Arithm. Circuits	×	A	✓	NA
[95]	Arithm. Circuits	×	A	×	×
[96]	Arithm. Circuits	×	A	×	×
[31]	Stateful	×	A	×	×
[94]	Arithm. Circuits	×	A	×	×
[103]	Arithm. Circuits	×	A	✓	NA
[86]	Arithm. Circuits	×	A	✓	I
[50]	Arithm. Circuits	×	A	✓	I
[18]	General Loops	×	A	✓	I
[19]	General Loops	×	A	✓	I
[11]	Arithm. Circuits	×	A	✓	I
[93]	Arithm. Circuits	I/O(inf.)	A	✓	I

Table 3.2: Proof-based verifiable computation schemes. Properties: privacy w.r.t. server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

We now discuss approaches to verifiable computing that use fully homomorphic encryption (FHE [66]) as a building block. For these solutions the client encrypts the data before it outsources it to the server. Thus, these solutions achieve input privacy. In addition, only the client can decrypt the result, which is why also output privacy is assured. However, on the downside all fully homomorphic encryption based schemes are only privately verifiable. Furthermore, all such solutions are only secure against weak adversaries as defined in Def. 2.33 and providing efficient FHE schemes is still an open research challenge.

Scheme	Function Class	$\mathcal{A}$	PrS	E	VER	PrV
[62]	Arithm. Circuits	W	I/O	A	×	NA
[47]	Arithm. Circuits	W	I/O	A	×	NA

Table 3.3: FHE-based verifiable computation schemes. Properties: adversary ($\mathcal{A}$), privacy w.r.t server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

Homomorphic authenticators allow to evaluate functions on *authenticated* data. There exist constructions both in the secret key setting in the form of *homomorphic message authentication codes (MACs)* and in the public key setting in the form of *homomorphic signatures*. Their properties are discussed in more detail in Section 2.2. These solutions can be used to respectively construct *privately* and *publicly verifiable*

computing schemes. There are homomorphic MAC and signature schemes that are not known to allow verification faster than computing the function, e.g. [65], and are therefore not considered here.

Scheme	Function Class	$\mathcal{A}$	PrS	Primitives	E	VER	PrV
[12]	Poly. of Degree 2	S	×	Bilinear Maps	A	×	×
[104]	Poly. of Fixed Degree	S	×	Multilinear Maps	A	×	×
[58]	Poly. of Degree 2	S	I	Bilinear Maps	A	×	×
[37]	Linear	S	×	Bilinear Maps	A	✓	I(inf.)
[41]	Poly. of Fixed Degree	S	×	Multilinear Maps	A	✓	NA
[36]	Poly. of Fixed Degree	S	×	RSA	A	✓	NA
[75]	D	D	I/O	HE/HEA [75]	D	D	D
[59]	Arithm. Circuits	S	×	Lattices	A	✓	×
[76]	D	S	×	SNARKs	A	✓	D

Table 3.4: Authenticator-based verifiable computation schemes. Properties: adversary ($\mathcal{A}$), privacy w.r.t server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

The schemes using homomorphic authentication, see Table 3.4, are more restrictive with respect to the supported function class. Furthermore, all schemes only provide amortized efficiency and only the solutions using homomorphic signature schemes provide public verifiability. The generic construction proposed by Lai et al. [75] allows to combine authentication based verifiability with encryption gaining a verifiable computing scheme preserving input-output privacy towards the server. Nevertheless, the function class, security, and efficiency depend on the underlying primitives and further research is required for identifying promising instantiations for different applications.

Another line of research are verifiable computing schemes based on functional encryption (FE) or functional signatures (FS), see Table 3.5. We present the verifiable computing schemes that use (key-policy) attribute-based encryption (ABE) or are directly constructed from functional encryption schemes. Key-policy ABE (KP-ABE) [72, 89] is a public key encryption paradigm, where a public key is associated to a universe of attributes A and secret keys are associated to boolean functions f. A holder of a secret key corresponding to f can only decrypt a message encrypted with respect to a subset A' of attributes iff $f(A') = 1$. FE [26] is a very generic definition of various types of public key encryption concepts, such as identity based encryption (IBE [22]), ABE, and many other classes. Basically, in such schemes secret keys are associated to a function f and given a ciphertext of a message m under the corresponding public key, the holder of a secret key corresponding to f will only learn $f(m)$ during decryption, instead of learning the full plaintext m. Assuming that the plaintext space has an additional structure and in particular plaintexts are pairs of some (public) index and message space,

then one can define FE on predicates over the index space and the key space. In doing so, one obtains KP-ABE as a so-called predicate encryption (PE) scheme with a public index. FS [28] is the signature analog to FE. It allows the delegated signing of a messages m if $f(m) = 1$ holds for a given function f.

Scheme	Function Class	$\mathcal{A}$	PrS	Primitives	E	VER	PrV
[87]	Boolean Functions	$\emptyset$	×	ABE	A	×	×
[13]	D	S	I/O	FE,MAC,FHE,PE	A	✓	D
[28]	Arithm. Circuits	W	×	FS	D	✓	NA

Table 3.5: FE- and FS-based verifiable computation schemes. Properties: adversary ($\mathcal{A}$), privacy w.r.t server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

Beyond the families of schemes we have seen so far, there exist verifiable computing schemes for specific functions, which we present in Table 3.6. These are fine-tailored to the verifiable computation of specific functions.

Scheme	Function Class	$\mathcal{A}$	PrS	Primitives	E	VER	PrV
[5]	Arithmetic Branching Programs	W	I	Randomized Encodings	A	×	×
[85]	Poly. + Derivations	S	×	Bilinear Maps	A	✓	NA
[56]	Univariate Poly.	S	×	Bilinear Maps	A	✓	NA
[56]	Matrix-Vector Multiplication	S	×	Bilinear Maps	A	✓	NA
[45]	Bilinear Maps	S	×	Bilinear Maps	A	×	×
[58]	Univariate Poly.	S	I	Bilinear Maps	A	×	×
[74]	Set Operations	S	×	SNARK	A	✓	NA
[20]	Poly. of Fixed Degree	S	×	Bilinear Map	A	×	×
[99]	Multivariate Polynomials	S	×	Bilinear Maps	A	✓	NA
[99]	Matrix-Vector Multiplication	S	×	Bilinear Maps	A	✓	NA

Table 3.6: Other verifiable computation schemes. Properties: adversary ($\mathcal{A}$), privacy w.r.t server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

There are several schemes that have been developed within this thesis. Those will be described in detail in the following chapters and compared to the state of the art.

4 Function-Dependent Commitment Schemes

When outsourcing computations, several security risks arise. One such risk are undetected incorrect results. Verifiability allows to detect incorrect results. Verifying a computation in less time than performing the computation itself gives obvious benefits for the client. Often, not only the data owner is interested in the correctness of a computation; but also third parties. Public verifiability allows such third party verifiers to check the correctness of an outsourced computation.

For sensitive data on which outsourced computation takes place, the leakage of private data is another security risk. In the case of publicly verifiable computing, this risk exists both with respect to the server(s) and the verifier. In this chapter, we show how to address these security risks. In order to achieve this, we present the fundamental building block of our information-theoretically private schemes — *function-dependent commitments (FDC)*. This novel cryptographic primitive is a generalization of both homomorphic authenticators and commitments. We furthermore present the first concrete instantiation of an FDC scheme. It allows the efficient public verification of linear functions. Finally, we show how to build an efficiently verifiable computing scheme for linear functions that provides complete information-theoretic privacy (information-theoretic input and output privacy with respect to both servers and verifiers), by combining our FDC scheme with linear secret sharing. A crucial security property of FDCs is the *hiding* property, which means a commitment to a message does not leak information about the message. This property can be used to achieve output privacy with respect to the verifier. This property furthermore allows us to achieve input and output privacy with respect to the servers when combining an FDC with secret sharing.

Contribution. In this chapter, we solve the problem of providing efficient verification with information-theoretic privacy for linear functions. To achieve this, we introduce a novel generic construction that combines information-theoretic privacy with strong unforgeability and fast verification. We call this construction *function-dependent commitments* (FDCs).

In addition, we provide a concrete, unconditionally hiding instantiation of FDCs for linear functions using pairings, demonstrating that our generic construction can be realized in the standard model. Finally, we showcase a verifiable multi-party computation scheme based on the concrete instantiation. This scheme makes it possible to verify whether the reconstructed result has been computed correctly by computing additional audit data on a *single* storage server. Previous proposals require *all* storage servers to perform computations to check correctness. Our scheme provides unconditional input-output privacy towards the servers and parties verifying computational correctness. In Figure 4.1 we show which properties of an FDC are used to address which particular challenge.

Organization. In this chapter we first introduce our framework for FDC schemes (Sec. 4.1). We then present a concrete instantiation of an FDC using pairings, and prove its properties (Sec. 4.2). Finally we show how this instantiation can be used to build a verifiable computing scheme for shared data (Sec. 4.3).

Publications. This chapter is based on publication [S2].

Related Work. FDCs combine properties of commitments (see Sec. 2.5) and homomorphic authenticators (see Sec. 2.2). In [79], the notion of *functional commitments* is introduced. Their notion of function bindingness, however, is strictly weaker than our notion of adaptive unforgeability. functional commitments. Functional commitments allow only a unique *opening* of a given commitment to the result of a linear computation. Our FDCs only allow the computation of a unique *commitment*.

Both our FDC and the homomorphic signature scheme presented in [37] are based on the FDHI assumption 2.42, and indeed our FDC builds on this homomorphic signature scheme. The Catalano–Fiore–Nizzardo construction is context hiding, i.e. signatures to the output of a function do not leak information about the inputs to the function beyond knowledge of the output to a third party verifier. By contrast, our FDC achieves an even stronger privacy property: information-theoretic input–output privacy with respect to both verifiers and servers. A freshly signed signature in the case of [37] still reveals information about the message to an adversary corrupting a server. Our unconditionally hiding FDC, however, does not. Furthermore, our verification algorithm FunctionVerify only requires a commitment to the output of a computation, enabling output privacy, while verification in [37] requires the output itself. This requires a novel strategy in our security reduction.

In [92] and [93] schemes for verifiable MPC are described. They demand that every shareholder performs computations in order to allow for verification and thereby produces a significant overhead. Our verifiable computing scheme for

shared data allows to process secret shares while only one storage server has to compute the audit data.

4.1 Function-Dependent Commitments

In this section, we present our novel framework of FDC schemes and define their relevant properties. We define the classical properties of commitments, binding and hiding, in the context of FDCs. Furthermore we provide definitions for evaluation correctness and unforgeability. In terms of performance properties, we consider succinctness and amortized efficiency.

Like in the case of homomorphic commitments or authenticators, a function-dependent commitment can be used to derive new commitments by its homomorphic properties. It is necessary that the homomorphic property cannot be abused to create forgeries. In the context of homomorphic authenticators, the notions of labeled and multi-labeled programs (see Section. 2.1) are introduced to provide meaningful security definitions.

Evaluating a function can be modeled as performing a program on a set of labeled inputs that belong to a given dataset. On a high level, a message is uniquely identified by two identifiers: one input identifier τ, and one dataset identifier Δ. One can think of a dataset as an array of message, and of the input identifiers as pointers to specific positions within this array.

This enables a precise description of homomorphic properties. For authenticators, it is usually required that only authenticators created under the same dataset identifier are used for homomorphic evaluation.

Using the formalism of multi-labeled programs as described in Section 2.1, we now define FDCs.

Definition 4.1. *A function dependent commitment (FDC) scheme for a class $\mathcal{F}$ of functions is a tuple of algorithms* (Setup, KeyGen, PublicCommit, PrivateCommit, FunctionCommit, Eval, FunctionVerify, PublicDecommit)*:*

- $\mathsf{Setup}(1^\lambda)$ *takes as input the security parameter λ and outputs public parameters* pp*. We implicitly assume that every algorithm uses these public parameters, leaving them out of the notation.*
- $\mathsf{KeyGen}(\mathsf{pp})$ *takes the public parameters* pp *as input and outputs a secret-public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*
- $\mathsf{PublicCommit}(m, r)$ *takes as input a message m and randomness r and outputs a commitment C and decommitment value d.*
- $\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$ *takes as input the secret key* sk*, a message m, randomness r, a dataset Δ, and an identifier τ and outputs an authenticator A for the tuple (m, r, Δ, τ).*

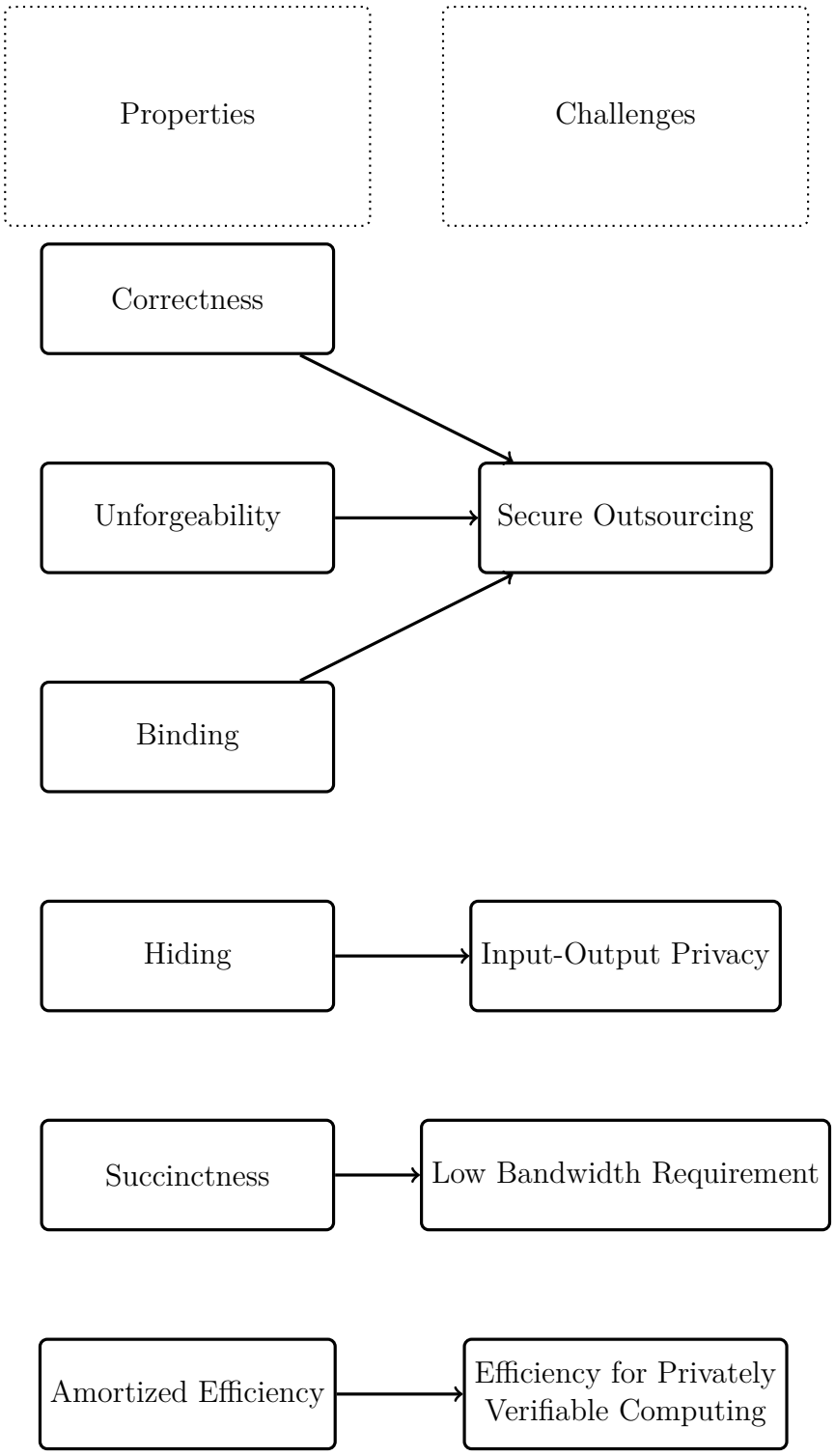

Figure 4.1: Properties of FDCs

$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$ *takes as input the public key* pk *and a labeled program* $\mathcal{P}$ *and outputs a function commitment* F *to* $\mathcal{P}$.

$\mathsf{Eval}(f, A_1, \ldots A_n)$ *takes as input a function* f *and a set of authenticators* $A_1, \ldots, A_n$*, where* A_i *is an authenticator for* $(m_i, r_i, \Delta, \tau_i)$*, for* $i = 1, \ldots, n$*. It outputs an authenticator* A^*.

$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$ *takes as input a public key* pk*, an authenticator* A*, a commitment* C*, a function commitment* F*, as well as a dataset identifier* Δ*. It outputs either 1 (accept) or 0 (reject).*

$\mathsf{PublicDecommit}(m, d, C)$ *takes as input message* m*, decommitment* d*, and commitment* C*. It outputs either 1 (accept) or 0 (reject).*

High Level Overview over FDCs FDCs allow for two different ways of committing to messages. One is just a standard commitment. This allows us to achieve output privacy with respect to the verifier. The other way commits to a message under a secret key to produce an authenticator. These authenticators allow for homomorphic evaluation. Given authenticators to the input of a function, we can derive an authenticator to the output of a function. We additionally allow to commit to a function under a public verification key. This results in a function commitment. Our scheme allows us to check if a public commitment C matches an authenticator A (derived from a secret key) and a function commitment F (derived from a public key). As long as a cryptographic hardness assumption holds, such a match is only possible if A was obtained by running the evaluation on the exact function committed to via F.

Regarding the unforgeability of FDCs, we face a fundamental problem in homomorphic cryptography. The homomorphic property allows for the creation of commitments not created by the original data owner. Our primitive seeks to limit this ability to avoid forgeries. To this end, we consider the messages we commit to as structured data. Messages are bundled in datasets and have a position within the datasets. This approach has been formalized under the notion of labeled programs (see e.g. [12]).

As for classical commitments, we want our schemes to be *binding*. That is, after committing to a message, it should be infeasible to open the commitment to a different message. We describe the following security experiment between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Definition 4.2 (Binding experiments)**.**

$\boldsymbol{EXP}^{Bind}_{\mathcal{A},Com}(\lambda)$:

Challenger $\mathcal{C}$ *runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and gives* pk *to the adversary* $\mathcal{A}$.

$\mathcal{A}$ *outputs a commitment* C *and the pairs* (m, d) *and* (m', d')*, with* $m \neq m'$.

If we have both $\mathsf{PublicDecommit}(m, d, C) = 1$ *and* $\mathsf{PublicDecommit}(m', d', C)$ *the experiment outputs '1', else it returns '0'.*

Definition 4.3 (Binding)**.** *Using the formalism of Def. 4.2, an FDC is called* binding *if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$,*

$$\Pr\Big[\boldsymbol{EXP}^{Bind}_{\mathcal{A},Com}(\lambda) = 1\Big] = \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ *denotes any function negligible in the security parameter* λ*.*

Note that defining the binding property for FunctionCommit works completely analogously.

We will also provide a weaker version of the binding property, the target binding property. As for ordinary commitments (see e.g. [81]) the difference is similar to the difference between collision resistance and second-preimage resistance for hash functions.

Definition 4.4 (Target bindingness experiments)**.**
$\boldsymbol{EXP}^{Target-Bind}_{\mathcal{A},Com}(\lambda)$:
Challenger $\mathcal{C}$ *runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and gives* pk *to the adversary* $\mathcal{A}$*.*
Challenger $\mathcal{C}$ *runs* $(C, d) \leftarrow \mathsf{PublicCommit}(m, r)$ *and gives* (C, d) *to* $\mathcal{A}$*.*
$\mathcal{A}$ *outputs a pair and* (m', d')*, with* $m \neq m'$*.*
If we have both $\mathsf{PublicDecommit}(m, d, C) = 1$ *and* $\mathsf{PublicDecommit}(m', d', C) = 1$
the experiment outputs '1', else it returns '0'.

Definition 4.5 (Target Binding)**.** *Using the formalism of Def. 4.4, an FDC is called* target binding *if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$,*

$$\Pr\Big[\boldsymbol{EXP}^{Target-Bind}_{\mathcal{A},Com}(\lambda) = 1\Big] = \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ *denotes any function negligible in the security parameter* λ*.*

Obviously the binding property implies the target binding property.

Another important notion, targeting privacy, is the hiding property. Commitments are intended not to leak information about the messages they commit to. This is not to be confused with the context hiding property, where homomorphic authenticators to the output of a computation do not leak information about the inputs to the computation. Context hiding homomorphic authenticators do however leak information about the output.

Definition 4.6 (Hiding)**.** *An FDC is called* computationally hiding *if the sets of commitments*

$$\{\mathsf{PublicCommit}(m, r) \mid r \xleftarrow{\$} \mathcal{R}\} \text{ and}$$

$$\{\mathsf{PublicCommit}(m', r') \mid r' \xleftarrow{\$} \mathcal{R}\}$$

as well as

$$\{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\} \text{ and}$$

$$\{\mathsf{PrivateCommit}(\mathsf{sk}, m', r', \Delta, \tau) \mid r' \xleftarrow{\$} \mathcal{R}\}$$

have distributions that are indistinguishable for any PPT adversary $\mathcal{A}$ for all $m \neq m' \in \mathcal{M}$.
An FDC is called unconditionally hiding *if these sets have the* same *distribution respectively for all $m \neq m' \in \mathcal{M}$.*

An obvious requirement for an FDC is to be *correct*, i.e. if messages are authenticated properly and evaluation is performed honestly, the resulting commitment should be verified. This is formalized in the following definition.

Definition 4.7 (Correctness)**.** *An FDC achieves* correctness *if for any security parameter λ, any public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and any dataset identifier $\Delta \in \{0,1\}^*$, the following properties hold:*

For any message $m \in \mathcal{M}$, any randomness $r \in \mathcal{R}$, any label $\tau \in \mathcal{T}$, and any authenticator $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$, commitment $C \leftarrow \mathsf{PublicCommit}(m, r)$ function commitment $F_{\mathcal{I}} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$, where $\mathcal{I}_\tau$ is the labeled identity program, we have

$$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F_{\mathcal{I}}, \Delta) = 1.$$

Let $g \in \mathcal{F}$ be any admissible function, $\{(A_i, m_i, r_i, \mathcal{P}_i)\}_{i\in[N]}$ be any tuple such that for $(C_i, d_i) \leftarrow \mathsf{PublicCommit}(m_i, r_i)$, $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$, $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$ the following holds: There exists a function $\hat{g} \in \mathcal{F}$, that is efficiently computable from g, such that for the following holds: Let $m^* = g(m_1, \ldots, m_N)$, $r^* = \hat{g}(m_1, \ldots, m_n, r_1, \ldots r_n)$, $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$, $A^* \leftarrow \mathsf{Eval}(g, A_1, \ldots, A_N)$. Then $\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F^*, \Delta) = 1$.

The security notion of FDCs is also based on *well defined programs* (see Def. 2.1). We introduce an experiment the attacker can run in order to generate a successful forgery and present a definition for unforgeability based on this experiment.

Definition 4.8 (Forgery)**.** *A forgery is a tuple $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$ such that*

$$\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*), \Delta^*) = 1$$

holds and exactly one of the following conditions is met:

Type 1 Forgery: *No message was ever committed under the data set identifier Δ^*, i.e. the list L_{Δ^*} of tuples (τ, m, r) was not initialized during the security experiment (see Def. 4.9).*

Type 2 Forgery: *$\mathcal{P}^*_{\Delta^*}$ is well defined with respect to list L_{Δ^*} and for $m^* = f(\{m_j\}_{(\tau_j,m_j,r_j)\in L_{\Delta^*}})$, $r^* = \hat{f}(\{d_j\}_{(\tau_j,m_j,r_j)\in L_{\Delta^*}}))$, where f is taken from $\mathcal{P}^*$, we have*

$$\mathsf{PublicDecommit}(C^*, m^*, r^*) = 0,$$

that is, C^* is not a commitment to the correct output of the computation.

Type 3 Forgery: *$\mathcal{P}^*_{\Delta^*}$ is not well defined with respect to L_{Δ^*}.*

To define unforgeability, we first describe the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Definition 4.9 ($\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda)$ [90])**.**

EXP$^{UF-CMA}_{\mathcal{A},FDC}(\lambda)$:

Setup *$\mathcal{C}$ calls $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and gives pp to $\mathcal{A}$.*

Key Generation *$\mathcal{C}$ calls $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$ and gives pk to $\mathcal{A}$.*

Queries *$\mathcal{A}$ adaptively submits queries for (Δ, τ, m, r) where Δ is a dataset, τ is an identifier, m is a message, and r is a random value. $\mathcal{C}$ proceeds as follows:*

- *If (Δ, τ, m, r) is the first query with dataset identifier Δ, it initializes an empty list $L_\Delta = \emptyset$ for Δ.*
- *If L_Δ does not contain a tuple $(\tau, \cdot, \cdot)$, that is, $\mathcal{A}$ never queried $(\Delta, \tau, \cdot, \cdot)$, $\mathcal{C}$ calls $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$, updates the list $L_\Delta = L_\Delta \cup (\tau, m, r)$, and gives A to $\mathcal{A}$.*
- *If $(\tau, m, r) \in L_\Delta$, then $\mathcal{C}$ returns the same authenticator A as before.*
- *If L_{Δ^*} already contains a tuple (τ, m', r') for $(m, r) \neq (m', r')$, $\mathcal{C}$ returns $\perp$.*

Forgery *$\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$.*

EXP$^{UF-CMA}_{\mathcal{A},FDC}(\lambda)$ *outputs '1' if the tuple returned by $\mathcal{A}$ is a forgery as defined before in Def. 4.8.*

Definition 4.10 (Unforgeability)**.** *An FDC is unforgeable if for any PPT adversary $\mathcal{A}$,*

$$\Pr\Big[\boldsymbol{EXP}^{UF-CMA}_{\mathcal{A},FDC}(\lambda) = 1\Big] = \mathsf{negl}(\lambda).$$

Regarding performance, we consider additional properties. *Succinctness* specifies a limit on the size of the FDCs, thus keeping the required bandwidth low when using FDCs to verify the correctness of an outsourced computation.

Definition 4.11 (Succinctness)**.** *An FDC is* succinct *if, for a fixed security parameter λ, the size of the authenticators depends at most logarithmically on the dataset size n.*

Amortized efficiency specifies a bound on the computational effort required to perform verifications.

Definition 4.12 (Amortized Efficiency)**.** *Let $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ be a multi-labeled program, $m_1, \ldots, m_n \in \mathcal{M}$ a set of messages, $r_1, \ldots r_n \in \mathcal{R}$ a set of randomness, $f \in \mathcal{F}$ be an arbitrary function, and $t(n)$ be the time required to compute $f(m_1, \ldots, m_n)$. An FDC achieves* amortized efficiency *if, for any public parameters* pp *and any* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$*, any authenticator A, any commitment C, and function commitment F, the time required to compute* $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$ *is $t' = o(t(n))$. Note that A and F may depend on f and n.*

Analogous definitions for amortized efficiency haven been given in [12], [37], and [41]. Compared to Def. 2.8 this definition does not require additional algorithms but rather has analogous requirements for the runtime of algorithms. The usual one-time pre-computation (VerPrep in Def. 2.8) is captured by our algorithm FunctionCommit and FunctionVerify is already required to provide efficient verification (analogous to EffVer in Def. 2.8). In the case of reuse of the same function over multiple datasets, this property enables an improvement in terms of runtime.

4.2 An FDC for Linear Functions

In the previous section, we introduced the idea of FDCs. In this section, we provide a concrete instantiation for linear functions that is succinct, supports efficient verification, and is information-theoretically hiding. We will prove these properties along with the basic properties of correctness and unforgeability. In terms of hardness assumptions, only a variant of the Diffie–Hellman problem [37] is required. We are now ready to describe the algorithms making up our FDC. We use a signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. For a set of possibly different messages $m_1, \ldots, m_n$, we denote by m_i the i-th message. Since our messages are vectors, i.e. $, \in \mathbb{F}_p^T$, we write $m[j]$ to indicate the j-th entry of message vector m. Therefore $m_i[j]$ denotes the j-th entry of the i-th message. Given a linear function f, its i-th coefficient is denoted by f_i. So we have $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$.

High level overview over our construction. Our FDC allows for two different ways of committing to messages. Both are derived from the linearly homomorphic Pedersen commitments [97]. We use this homomorphic property as the basis of our public evaluation algorithm Eval. However, in order to achieve unforgeability further steps are necessary. Our authenticators also depend on the identifiers used to label the messages as structured data. Both the secret key and the verification

key contain elements associated to the identifiers. This allows us to link the secret elements used when creating an authenticator to the public elements used in creating a function commitment.

Construction 4.13.

$\mathsf{Setup}(1^\lambda)$*: On input security parameter* λ*, the algorithm runs* $\mathcal{G}(1^\lambda)$ *to obtain a bilinear group* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$*. It chooses* $m, T \in \mathbb{N}$*. Then it chooses* $H_0, \ldots, H_T \xleftarrow{\$} \mathbb{G}_1$ *uniformly at random. It checks whether* $H_j \neq H_i$ *for all* $j \neq i$*. If not it chooses new* H_j*. Note that this check fails only with negligible probability. Additionally, it fixes a regular signature scheme* $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ *and a a pseudorandom function* $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$*. It outputs the public parameters* $\mathsf{pp} = (\lambda, n, T, \mathsf{bgp}, H_0, \ldots H_T, \mathsf{Sig}, \Phi)$*.*

$\mathsf{KeyGen}(\mathsf{pp})$ *: On input public parameters* pp *it chooses* $y \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random and computes* $Y = g_2^y$*. Additionally it chooses* $b_1, \ldots b_n \xleftarrow{\$} \mathbb{F}_p$ *uniformly at random and checks whether* $b_i \neq b_j$ *for all* $i \neq j$*. If it fails it chooses a new* b_i*. Note that this check is failed only with negligible probability. Then it computes* $h_i = g_t^{b_i}$ *for all* $i \in [n]$*. Additionally the algorithm chooses random seeds* $K, K' \xleftarrow{\$} \mathcal{K}$ *for the pseudorandom function* Φ*. It computes keys for the regular signature scheme* $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*. It sets* $\mathsf{sk} = (b_1, \ldots, b_n, \mathsf{sk}_{\mathsf{Sig}}, K, K', y)$ *and* $\mathsf{pk} = (h_1, \ldots, h_n, \mathsf{pk}_{\mathsf{Sig}}, Y)$*. It outputs* $(\mathsf{sk}, \mathsf{pk})$*.*

$\mathsf{PublicCommit}(m, r)$*: On input a message* $m \in \mathbb{F}_p^T$ *and randomness* $r \in \mathbb{F}_p$ *it computes* $C = H_0^r \cdot \prod_{j=1}^T H_j^{m[j]}$*, where* $m[j]$ *is the* j*-th entry of message vector* $m \in \mathbb{F}_p^T$*, and outputs the commitment* C *and decommitment value* $d = r$*.*

$\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$*: On input a secret key* sk*, a message* $m \in \mathbb{F}_p^T$*, randomness* $r \in \mathbb{F}_p$*, a dataset* $\Delta \in \{0,1\}^*$*, and an identifier* $\tau \in [n]$ *the algorithm first computes* $z = \Phi_K(\Delta)$ *and calculates* $Z = g_2^z$*. Then, it binds* Z *to the dataset identifier* Δ *by signing their concatenation, i.e.*$\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta || Z)$*. Then, it sets* $u = \Phi_{K'}(\Delta || \tau)$*,* $U = g_1^u$*, and* $\Lambda = \left(U \cdot g_1^{b_\tau} \cdot H_0^{yr} \cdot \prod_{j=1}^T H_j^{ym[j]}\right)^{\frac{1}{z}}$*. It returns the authenticator* $A = (\sigma_\Delta, Z, U, \Lambda)$*.*

$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$*: On input a public key* pk *and a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*. It computes* $F = \prod_{i=1}^n h_i^{f_i}$*, where* f_i *denotes the* i*-th coefficient of* f*, and outputs the function commitment* F*.*

$\mathsf{Eval}(f, A_1, \ldots A_n)$*: On input a linear function* f *and authenticators* $A_1, \ldots, A_n$ *the algorithm parses* $A_i = (\sigma_{\Delta,i}, Z_i, U_i, \Lambda_i)$*. It sets* $\sigma_\Delta = \sigma_{\Delta,1}$*,* $Z = Z_1$ *and computes* $U = \prod_{i=1}^n U_i^{f_i}$ *and* $\Lambda = \prod_{i=1}^n \Lambda_i^{f_i}$ *and outputs authenticator* $A = (\sigma_\Delta, Z, U, \Lambda)$*.*

$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$*: On input a public key* pk*, an authenticator* A*, a commitment* C*, a function commitment* F*, and a dataset identifier* Δ*, the*

algorithm parses $A = (\sigma_\Delta, Z, U, \Lambda)$ *It checks whether* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$ *holds. If not it outputs '0', otherwise it checks whether* $e(\Lambda, Z) = e(U, g_2) \cdot F \cdot e(C, Y)$ *holds. If it does, it outputs '1'; otherwise it outputs '0'.*

$\mathsf{PublicDecommit}(m, d, C)$*: On input a message* $m \in \mathbb{F}_p^T$*, decommitment* $d \in \mathbb{F}_p$*, and a commitment* C*. It outputs '1'if* $C = \mathsf{PublicCommit}(m, d)$ *and '0'otherwise.*

Our construction can also be used to provide authenticity in the form of unconditionally hiding authenticators, similarly to signatures. First, algorithm KeyGen is called. To authenticate a message m, the owner of the secret key sk generates a random value r and computes an authenticator A with algorithm $\mathsf{PrivateCommit}$. The authenticator A serves as a signature for m. To verify the authenticity of m, the verifier computes the commitment C by calling $\mathsf{PublicCommit}(m, r)$. Then, it uses pk and algorithm $\mathsf{FunctionCommit}$ to generate a function commitment $F_{\mathcal{ID}}$ to the identity function. Finally, it calls algorithm $\mathsf{FunctionVerify}$ to check whether the tuple $C, A, F_{\mathcal{ID}}, \Delta$ is valid.

In the appendix we present the runtimes of Construction 4.13 in Sec. 4.2.

In the following, we first prove that our concrete scheme is indeed correct in the sense of Def.4.7. We then prove that it satisfies the classical commitment properties — binding and hiding. With respect to efficiency, we next show succinctness and amortized efficiency. Finally, we reduce the security of our scheme to the hardness of the FDHI assumption (see Def. 2.42).

Theorem 4.14. *Our construction is a correct FDC in the sense of Def. 4.7 if* Sig *is a correct signature scheme.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ arbitrary public parameters, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ be an arbitrary key pair, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier.

Now let $m \in \mathbb{Z}_p^T$ be an arbitrary message, $r \in \mathbb{Z}_p$ be arbitrary randomness, and $\tau \in [n]$ be an arbitrary input identifier, as well as $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$, $C \leftarrow \mathsf{PublicCommit}(m, r)$, and $F_{\mathcal{I}} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$, where $\mathcal{I}_\tau$ is the labeled identity program. Then we have $A = (\sigma_\Delta, Z, U, \Lambda)$ with $\sigma_\Delta = \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$. It computes $u = \Phi_{K'}(\Delta||\tau)$ and $\Lambda = \left(U \cdot g_1^{b_\tau} \cdot H_0^{yr} \cdot \prod_{j=1}^T H_j^{ym[j]}\right)^{\frac{1}{z}}$. If Sig is a correct signature scheme we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$.

Furthermore we have

$$
\begin{aligned}
e(\Lambda, Z) &= e\left(\left(U \cdot g_1^{b_\tau} \cdot H_0^{yr} \cdot \prod_{j=1}^T H_j^{ym[j]}\right)^{\frac{1}{z}}, g_2^z\right) \\
&= e\left(U \cdot g_1^{b_\tau} \cdot H_0^{yr} \cdot \prod_{j=1}^T H_j^{ym[j]}, g_2\right)
\end{aligned}
$$

$$= e\left(U, g_2\right) \cdot e\left(g_1^{b_\tau}, g_2\right) \cdot e\left(H_0^r \cdot \prod_{j=1}^{T} H_j^{m[j]}, g_2^y\right)$$
$$= e\left(U, g_2\right) \cdot F_{\mathcal{I}} \cdot e\left(C, Y\right)$$

Therefore all checks of $\mathsf{FunctionVerify}$ pass.

Now consider a set of arbitrary tuples $\{(A_i, m_i, r_i, \mathcal{P}_i)\}_{i\in[N]}$ such that for $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$, $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$, we have $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$.

Let $m^* = g(m_1, \ldots, m_N) = \sum_{i=1}^{N} g_i m_i$, $r^* = \hat{g}(m_1, \ldots, m_n, r_1, \ldots r_n) = \sum_{i=1}^{N} g_i r_i$, $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$, $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_N)$.

We parse $A^* = (\sigma_\Delta^*, Z^*, \Lambda^*, U^*)$

Then we have by construction $\sigma_\Delta^* = \sigma_{\Delta,1}$ and $Z^* = Z_i$ for all $i \in [N]$ and therefore $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$.

Note that $F^* = \prod_{i=1}^{N} F_i^{g_i}$ by construction.

We also have

$$e\left(\Lambda^*, Z^*\right) = e\left(\prod_{i=1}^{N} \Lambda_i^{g_i}, Z^*\right) = \prod_{i=1}^{N} e\left(\Lambda_i, Z_i\right)^{g_i} = \prod_{i=1}^{N} \left(e\left(U_i, g_2\right) \cdot F_i \cdot e\left(C_i, Y\right)\right)^{g_i}$$
$$= e\left(\prod_{i=1}^{N} U_i^{g_i}, g_2\right) \cdot \prod_{i=1}^{N} F_i^{g_i} \cdot e\left(\prod_{i=1}^{N} C_i^{g_i}, Y\right)$$
$$= e\left(U^*, g_2\right) \cdot F^* \cdot e\left(\prod_{i=1}^{N} \left(H_0^{r_i} \cdot \prod_{j=1}^{T} H_j^{m_i[j]}\right)^{g_i}, Y\right)$$
$$= e\left(U^*, g_2\right) \cdot F^* \cdot e\left(H_0^{\sum_{i=1}^{N} g_i r_i} \cdot \prod_{j=1}^{T} H_j^{\sum_{i=1}^{N} g_i m_i[j]}, Y\right)$$
$$= e\left(U^*, g_2\right) \cdot F^* \cdot e\left(H_0^{r^*} \cdot \prod_{j=1}^{T} H_j^{\sum_{i=1}^{N} g_i m^*[j]}, Y\right)$$
$$= e\left(U^*, g_2\right) \cdot F^* \cdot e\left(C^*, Y\right)$$

Therefore all checks of $\mathsf{FunctionVerify}$ pass. This shows the correctness of our scheme.

□

The binding property of a commitment ensures that is can only be opened to one message. This property enables the use of FDCs for verifiable computing, as

our algorithm FunctionVerify only verifies a commitment C. If this is binding we know that this can only be opened to one specific message, the correct result of the computation.

Theorem 4.15. *Construction 4.13 is a binding FDC scheme in the sense of Def.4.3 as long as the discrete logarithm problem (see Def. 2.36) in $\mathbb{G}_1$ is hard.*

Proof. The following proof is adapted from [29, Section 2.3.2].

Assume we have a PPT adversary $\mathcal{A}$ that can break the binding property of our construction. We will show how a simulator $\mathcal{S}$ can use this to break the DL problem in $\mathbb{G}_1$. It takes as input $(\mathsf{bgp}, h_1 \in \mathbb{G}_1)$.

Simulator $\mathcal{S}$ chooses $a_j, b_j \xleftarrow{\$} \mathbb{F}_p$ uniformly at random for $j \in [T]$ and sets $H_j = g_1^{a_j} h_1^{b_j}$ for $j \in [T]$, as well as $H_0 = g_1$. $\mathcal{S}$ gives $H_0, \dots H_T$ to the adversary $\mathcal{A}$. Since the a_j and b_j are chosen uniformly at random these H_j are perfectly indistinguishable from elements sampled uniformly at random in $\mathbb{G}_1$. Let $(m, r), (m', r')$ be the output of the adversary. If this breaks the binding property then we have $m \neq m'$ and $H_0^r \cdot \prod_{j=1}^T H_j^{m[j]} = H_0^{r'} \cdot \prod_{j=1}^T H_j^{m'[j]}$.

$\mathcal{S}$ sets $a = (r' - r) + \sum_{j=1}^T a_j(m'[j] - m[j])$, $b = \sum_{j=1}^T b_j(m[j] - m'[j])$. If $b = 0$, it aborts. Otherwise, it outputs $x = \frac{a}{b} = \frac{(r'-r)+\sum_{j=1}^T a_j(m'[j]-m[j])}{\sum_{j=1}^T b_j(m[j]-m'[j])}$. Since we have

$$
\begin{aligned}
& H_0^r \cdot \prod_{j=1}^T H_j^{m[j]} = H_0^{r'} \cdot \prod_{j=1}^T H_j^{m'[j]} \Leftrightarrow H_0^{r-r'} \cdot \prod_{j=1}^T H_j^{(m[j]-m'[j])} = 1 \\
& \Leftrightarrow g_1^{r-r'} \cdot \left(g_1^{a_j} h_1^{b_j}\right)^{(m[j]-m'[j])} = 1 \\
& \Leftrightarrow h_1^{\sum_{j=1}^T b_j(m[j]-m'[j])} = g_1^{(r'-r)+\sum_{j=1}^T a_j(m'[j]-m[j])} \\
& \Leftrightarrow h_1 = g_1^x
\end{aligned}
$$

This is a solution to the DL problem. The b_j are information-theoretically hidden from the adversary $\mathcal{A}$, and are thus independent of $\mathcal{A}$'s output. Since $m^* \neq \hat{m}$, there exists a $j \in [T]$ such that $(m[j] - m'[j]) \neq 0$. For any tuple $(b_1, \dots, b_{j-1}, b_{j+1}, \dots b_T) \in \mathbb{F}_p^{T-1}$, there exists a unique $b_j \in \mathbb{F}_p$ such that $b = \sum_{j=1}^T b_j(m[j] - m'[j]) = 0$. Since all b_j are chosen uniformly at random, the probability that $\mathcal{S}$ aborts is therefore $\frac{p^{T-1}}{p^T} = \frac{1}{p}$.

The binding property for FunctionCommit works completely analogously. □

The hiding property will be the key to achieving information-theoretic privacy. On the one hand it ensures privacy with respect to the server as not even a computationally unbounded adversary can derive information about inputs to a computation from the authenticators to these inputs. Note that this is merely a

necessary condition for achieving information-theoretic input and output privacy with respect to the servers. We will combine our FDC with secret sharing to actually achieve it. On the other hand the hiding property is used to obtain information-theoretic output privacy with respect to the verifier. A (third party) verifier can check the correctness of a computation without even having to learn the result, as only an unconditionally hiding commitment is used during verification. This is captured in the following Theorem.

Theorem 4.16. *Construction 4.13 is an unconditionally hiding FDC in the sense of Def. 4.6.*

Proof. If $r \xleftarrow{\$} \mathbb{Z}_p$ is chosen uniformly at random then $\{H_0^r \mid r \xleftarrow{\$} \mathbb{Z}_p\}$ is uniformly distributed over $\mathbb{G}_1$. Therefore the set $\{H_0^r \cdot \prod_{j=1}^{T} H_j^{m[j]} \mid r \xleftarrow{\$} \mathbb{Z}_p\}$ is uniformly distributed over $\mathbb{G}_1$ for every $m \in \mathbb{Z}_p^T$. So $\{\mathsf{PublicCommit}(m, r) \mid r \xleftarrow{\$} \mathbb{Z}_p\}$ and $\{\mathsf{PublicCommit}(m', r') \mid r' \xleftarrow{\$} \mathbb{Z}_p\}$ have the same distribution $\forall m, m' \in \mathbb{Z}_p^T$. The output of $\mathsf{PrivateCommit}$ is an authenticator $A = (\sigma_\Delta, Z, U, \Lambda)$. By construction σ, Z, U are all independent of m. Considering the Λ component, we have $\Lambda = \left(U \cdot h_\tau \cdot \left(H_0^r \cdot \prod_{j=1}^{T} H_j^{m[j]}\right)^y\right)^{\frac{1}{z}}$. This is uniformly distributed over $\mathbb{G}_1$ if and only if the set $\{H_0^r \cdot \prod_{j=1}^{T} H_j^{m[j]} \mid r \xleftarrow{\$} \mathbb{Z}_p\}$ is. As we have shown this to be true $\{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \in \mathbb{Z}_p\}$ and $\{\mathsf{PrivateCommit}(\mathsf{sk}, m', r', \Delta, \tau) \mid r' \in \mathbb{Z}_p\}$ have the same distribution for all $m, m' \in \mathbb{Z}_p^T$.

□

We will now look at the efficiency properties of our construction. On the one hand we show its low bandwidth requirements in the from of succinctness. On the other hand we show that our verification procedure $\mathsf{FunctionVerify}$ runs in constant time.

Proposition 4.17. *Construction 4.13 is succinct in the sense of Def. 4.11.*

Proof. An authenticator produced by either $\mathsf{PrivateCommit}$ or Eval consists of a conventional signature, two elements in $\mathbb{G}_1$ and one element in $\mathbb{G}_2$. This is a constant and independent of n. Therefore our construction is succinct. □

Proposition 4.18. *Construction 4.13 achieves amortized efficiency in the sense of Def. 4.12.*

Proof. Let $t(n)$ be the running time of an evaluation of the linear function f. Then we have $t(n) = \mathcal{O}(n)$. An evaluation of $\mathsf{FunctionVerify}$ consists of a signature verification, two pairing evaluations, and two group operations. Thus their combined running time is independent of n. Therefore, our construction achieves amortized efficiency for suitably large n. □

Implementation

We now report on the experimental results of a Rust implementations of Construction 5.3. The measurements are based on an implementation by Rune Fiedler and Lennart Braun. As a pairing group the BLS curve [15] *BLS12-381* [27] is used.

The following measurements were executed on an Intel Core i7-4770K (Haswell) processor running at 3.50 GHz.

We present the runtimes of the individual subalgorithms of the FDC presented in Construction 4.13. We first present the runtimes influenced by the dimension T of vectors $m \in \mathbb{Z}_p^T$ given as messages in Table 4.1 and then present the runtimes influenced by the number of inputs n messages in Table 4.2.

Dimension	Setup	PublicCommit	PrivateCommit
32	9734	9667	11442
64	19176	19018	20768
128	38065	37913	39821
256	75835	75467	77344
512	151475	150746	152616

Table 4.1: Runtimes of our FDC 4.13 in μs

Inputs	KeyGen	FunctionCommit	Eval	FunctionVerify
256	525987	448335	151895	6972
512	1049245	896659	303786	6974
1024	2096320	1795722	608621	6977
2048	4191057	3588564	1216129	6976
4096	8380399	7175238	2431693	6976

Table 4.2: Runtimes of our FDC 4.13 in μs

We now address the issue of unforgeability and provide a security reduction of our FDC scheme.

Theorem 4.19. *Our construction is an unforgeable FDC scheme in the sense of Def. 4.10 if* Sig *is an unforgeable signature scheme,* Φ *is a pseudorandom function, and the FDHI assumption (see Def. 2.42) holds in* bgp*.*

Proof. This proof follows the structure of [37, Theorem 8]. A major difference is that, in our security reduction, the actual outcome of the computation function f is never required. In particular [38, Lemmata 5 and 7], knowledge of the forged outcome of the computation is a crucial part of the security reductions that prove

indistinguishability between games. We present a new indistinguishability reduction that only uses group elements.

To prove Theorem 4.19, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1' only with negligible probability. Following the notation of [37], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We use flag values bad_i, initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during game i.

It is an immediate corollary of Proposition 2.16, that any adversary who outputs a Type 3 forgery (see Def. 4.8) can be converted into one that outputs a Type 2 forgery. Therefore we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where $\mathcal{A}$ makes no corruption queries and only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 1, except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, r^*, A^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 4.20, we show that Bad_2 cannot occur if Sig is unforgeable. It is worth noticing that after this change the game never outputs '1'if $\mathcal{A}$ returns a Type 1 forgery.

Game 3 is defined as Game 2, except that the keyed pseudorandom function Φ_K is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. In Lemma 4.21 we show that these two games are indistinguishable if Φ is pseudorandom.

Game 4 is defined as Game 3, except except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_4 = \mathsf{true}$. In Lemma 4.22 we show that $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Game 5 is defined as Game 4, except for the following change. At the beginning, $\mathcal{C}$ chooses $z_\mu \in \mathbb{Z}_p$ at random and computes $Z_\mu = g_2^{z_\mu}$. It uses Z_μ whenever queried for dataset Δ_μ. It chooses $b_i, s_i \in \mathbb{Z}_p$ uniformly at random for $i = 1, \ldots n$ and sets $h_i = g_t^{b_i + z_\mu s_i}$. If $k = \mu$, simulator $\mathcal{S}$ sets the component $U_\tau = g_1^{-b_\tau - a_0 yr - \sum_{j=1}^{T} a_j y m[j]}$. In Lemma 4.23, we show that $\Pr[G_5(\mathcal{A})] = \Pr[G_4(\mathcal{A})]$.

Game 6 is defined as Game 5, except for the following change. The challenger runs an additional check. It computes $\hat{m} = f(m_1, \ldots, m_n), \hat{r} = f(r_1, \ldots, r_n)$, as well as $\hat{A} = \mathsf{Eval}(f, A_1, \ldots, A_n)$, i.e. it runs an honest computation over the

messages, randomness and authenticators in dataset Δ_μ. If

$$\mathsf{FunctionVerify}(\mathsf{vk}, A^*, C^*, \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*), \Delta^*)) = 1$$

and $U^* = \hat{U}$, then $\mathcal{C}$ sets $\mathsf{bad}_6 = \mathsf{true}$ In Lemma 4.24, we show that any adversary $\mathcal{A}$ for which Bad_6 occurs implies a solver for the FDHI problem.

Game 7 is defined as Game 6, except for the following change. During a query for (Δ_μ, τ, m, r), the challenger sets $U_\tau = g_1^{-b_\tau}$. In Lemma 4.25, we show that $\Pr[G_7(\mathcal{A})] = \Pr[G_6(\mathcal{A})]$. Finally in Lemma 4.26, we show that any adversary $\mathcal{A}$ that wins Game 7 implies a solver for the FDHI problem.

□

Lemma 4.20. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_2(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. Games 2 and 1 only differ if Bad_2 occurs, i.e. the list L_{Δ^*} was never initialized during the security experiment. In case of a successful forgery this means that there are valid signatures $\sigma_{\Delta^*,\mathsf{id}}$ for the concatenation $(\Delta^*||Z_{\mathsf{id}})$. even though no signature on $(\Delta^*||\cdot)$ was ever generated by the challenger. This immediately leads to an existential forgery for the regular signature scheme Sig, i.e. $\Pr[\mathsf{Bad}_2] = |\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig,F}}(\lambda)$. □

Lemma 4.21. *For every PPT adversary $\mathcal{A}$ running Game 3, there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{\Phi,\mathsf{D}}(\lambda)$.

Proof. Assume we have a noticeable difference $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \geq \epsilon$. Since the only difference between these games is the replacement of the pseudorandom function Φ by the random function $\mathcal{R}$, this immediately leads to a distinguisher $\mathcal{D}$ that achieves an advantage of ϵ against the pseudorandomness of Φ. □

Lemma 4.22. *For every PPT adversary $\mathcal{A}$ running Game 4, we have* $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Proof. First, $\Pr[G_3(\mathcal{A})] = \Pr[G_4(\mathcal{A}) \wedge \mathsf{bad}_4 = \mathsf{false}] = \Pr[G_4(\mathcal{A}) \mid \mathsf{bad}_4 = \mathsf{false}] \cdot \Pr[\mathsf{bad}_4 = \mathsf{false}]$, since Game 4 will always output '0' when Bad_4 occurs. Second, observe that when Bad_4 does not occur, i.e. $\mathsf{bad}_4 = \mathsf{false}$, the challenger guessed the dataset Δ^* correctly and the outcome of Game 4 is identical to the outcome of Game 3. Since μ is chosen uniformly at random and is completely hidden to $\mathcal{A}$, we have $\Pr[\mathsf{bad}_4 = \mathsf{false}] = \frac{1}{Q}$ and therefore $\Pr[G_4(\mathcal{A})] = \frac{1}{Q}\Pr[G_3(\mathcal{A})]$. □

Lemma 4.23. *We have* $\Pr[G_5(\mathcal{A})] = \Pr[G_4(\mathcal{A})]$

Proof. The two games are perfectly indistinguishable, corresponding to two different samplings of randomness. □

Lemma 4.24. *If there exists a PPT adversary $\mathcal{A}$ for whom Bad_6 occurs with non-negligible probability during Game 7 as described in Theorem 4.19, there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Definition 2.42) with non-negligible probability.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce the result Bad_6 during Game 6. We show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^u, g_1^{\frac{u}{v}})$, simulator $\mathcal{S}$ simulates Game 6.

Setup : Simulator $\mathcal{S}$ chooses $a_j \in \mathbb{Z}_p$ uniformly at random for $j \in [T]$ and sets $H_j = g_1^{a_j}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. It gives the public parameters $\mathsf{pp} = (\lambda, n, T, \mathsf{bgp}, H_0, \dots H_T, \mathsf{Sig}, \mathcal{R})$ to $\mathcal{A}$.

KeyGen : Simulator $\mathcal{S}$ chooses an index $\mu \in \{1, \dots, Q\}$ uniformly at random. During key generation, it chooses $b_i, s_i \in \mathbb{Z}_p$ uniformly at random for all $i = 1, \dots, n$. It sets $h_i = g_t^{b_i} \cdot e(g_1, g_2^z)^{s_i}$. It chooses $y \in \mathbb{Z}_p$ uniformly at random and sets $Y = g_2^y$. Additionally, it generates a key pair $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It gives the public key $\mathsf{pk} = (h_1, \dots, h_n, \mathsf{pk}_{\mathsf{Sig}}, Y)$ to $\mathcal{A}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m, r), which collects all the label/message/randomness tuples queried by the adversary on Δ and the respectively generated authenticators. Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_\mu \in \mathbb{Z}_p$, sets $Z_\mu = (g_2^z)^{\xi_\mu}$ and stores ξ_μ. If $k \neq \mu$, it samples a random $\xi_k \in \mathbb{Z}_p$ and sets $Z_k = (g_2^v)^{\xi_k}$ and stores ξ_k. Since all Z_k are randomly distributed in $\mathbb{G}_2$ they have the same distribution as in Game 6. Given a query (Δ, τ, m, r) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_k \mid\mid Z_k)$.

If $k \neq \mu$, it samples $\rho_\tau \in \mathbb{Z}_p$ uniformly at random and computes $U_\tau = g_1^{-b_\tau} \cdot (g_1^u)^{\rho_\tau} \cdot g_1^{-a_0 yr - \sum_{j=1}^{T} a_j y m[j]}$, $\Lambda_\tau = \left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_{tau}}\right)^{\frac{1}{\xi_k}}$, and gives $A =$

$(\sigma_{\Delta_k}, Z_k, U_\tau, \Lambda_\tau)$ to $\mathcal{A}$. We have

$$
\begin{aligned}
e(\Lambda_\tau, Z_k) &= e\left(\left(\left(g_1^{\frac{z}{v}}\right)^{s_\tau} \cdot \left(g_1^{\frac{u}{v}}\right)^{\rho_\tau}\right)^{\frac{1}{\xi_k}}, Z_k\right) \\
&= e\left(((g_1^z)^{s_\tau} \cdot (g_1^u)^{\rho_\tau})^{\frac{1}{v\xi_k}}, Z_k\right) \\
&= e\left(\left((g_1^z)^{s_\tau} \cdot g_1^{b_\tau - b_\tau} \cdot g_1^{-a_0 yr - \sum_{j=1}^{T} a_j ym[j] + a_0 yr + \sum_{j=1}^{T} a_j ym[j]} \cdot (g_1^u)^{\rho_\tau}\right)^{\frac{1}{z_k}}, Z_k\right) \\
&= h_\tau \cdot e(U_\tau, g_2) \cdot e\left(g_1^{a_0 r} \prod_{j=1}^{T} g_1^{a_j m[j]}, g_2^y\right) \\
&= h_\tau \cdot e(U_\tau, g_2) \cdot e(\mathsf{PublicCommit}(m, r), Y)
\end{aligned}
$$

This output is indistinguishable from the challenger's output during Game 6.

If $k = \mu$, simulator $\mathcal{S}$ sets $U_\tau = g_1^{-b_\tau - a_0 yr - \sum_{j=1}^{T} a_j ym[j]}$, computes $\Lambda_\tau = (g_1^{s_\tau})^{\frac{1}{\xi_\mu}}$ and gives $A = (\sigma_{\Delta_\mu}, Z_\mu, U_\tau, \Lambda_\tau)$ to $\mathcal{A}$. We have

$$
\begin{aligned}
&e(\Lambda_\tau, Z_\mu) = e\left((g_1^{s_\tau})^{\frac{1}{\xi_\mu}}, Z_\mu\right) = e\left((g_1^{zs_\tau})^{\frac{1}{z\xi_\mu}}, Z_\mu\right) = e\left((g_1^{zs_\tau})^{\frac{1}{z_\mu}}, Z_\mu\right) \\
&= e\left((g_1^{zs_\tau} \cdot g_1^{b_\tau - b_\tau + a_0 yr + \sum_{j=1}^{T} a_j ym[j] - a_0 yr - \sum_{j=1}^{T} a_j ym[j]})^{\frac{1}{z_\mu}}, Z_\mu\right) \\
&= e\left(g_1^{zs_\tau} \cdot g_1^{b_\tau - b_\tau + a_0 yr + \sum_{j=1}^{T} a_j ym[j] - a_0 yr - \sum_{j=1}^{T} a_j ym[j]}, g_2\right) \\
&= g_t^{zs_\tau} \cdot g_t^{-b_{tau}} \cdot g_t^{b_\tau} \cdot g_t^{ya_0 r + \sum_{j=1}^{T} ya_j m[j]} = h_\tau \cdot e(U_\tau, g_2) \cdot e(\mathsf{PublicCommit}(m, r), Y).
\end{aligned}
$$

Thus this output is indistinguishable from the challenger's output during Game 6.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, C^*, A^*)$ with $A^* = (\sigma^*_{\Delta^*}, Z^*, U^*, \Lambda^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 6 to compute $\hat{U}, \hat{\Lambda}, \hat{C} = \mathsf{PublicCommit}(\hat{m}, \hat{r})$. If Bad_6 occurs, we have $e(\Lambda^*, Z_\mu) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e(U^*, g_2) \cdot e(C^*, Y)$ and $e\left(\hat{\Lambda}, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(\hat{U}, g_2\right) \cdot e\left(\hat{C}, Y\right)$ Dividing those equations and using the fact that $\hat{U} = U^*$ we obtain $\frac{\Lambda^*}{\hat{\Lambda}} = (\frac{C^*}{\hat{C}})^{\frac{y}{z\xi_\mu}}$ or equivalently $(\frac{\Lambda^*}{\hat{\Lambda}})^{\xi_\mu} = (\frac{C^*}{\hat{C}})^{\frac{y}{z}}$ and therefore $W = (\frac{C^*}{\hat{C}})^y$ and $W' = (\frac{\Lambda^*}{\hat{\Lambda}})^{\xi_\mu}$ are a solution to the FDHI problem. By the definition of unforgeability, we have $W \neq 1$.

□

Lemma 4.25. *We have* $\Pr[G_7(\mathcal{A})] = \Pr[G_6(\mathcal{A})]$

Proof. The two games are perfectly indistinguishable, corresponding two different samplings of randomness. □

Lemma 4.26. *Assume there exists a PPT adversary $\mathcal{A}$ who wins Game 7 with non-negligible probability. Then there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Def. 2.42) with non-negligible probability.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that wins Game 7. We show how a simulator $\mathcal{S}$ can use this to solve the FDHI problem.

Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^u, g_1^{\frac{r}{u}})$, simulator $\mathcal{S}$ simulates Game 7.

Setup : Simulator $\mathcal{S}$ chooses $a_j \in \mathbb{Z}_p$ uniformly at random for $j \in [T]$ and sets $H_j = g_1^{a_j}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. It gives the public parameters $\mathsf{pp} = (\lambda, n, T, \mathsf{bgp}, H_0, \dots H_T, \mathsf{Sig}, \mathcal{R})$ to $\mathcal{A}$.

KeyGen : Simulator $\mathcal{S}$ chooses an index $\mu \in \{1, \dots, Q\}$ uniformly at random. During key generation, it chooses $b_i, s_i \in \mathbb{Z}_p$ uniformly at random for all $i = 1, \dots, n$. It sets $h_i = g_t^{b_i} \cdot e(g_1, g_2^z)^{s_i}$. It sets $Y = g_2^z$. Additionally, it generates a key pair $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It gives the public key $\mathsf{pk} = (h_1, \dots, h_n, \mathsf{pk}_{\mathsf{Sig}}, Y)$ to $\mathcal{A}$.

Queries: Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m, r), which collects all label/message/randomness tuples queried by the adversary on Δ and the respectively generated authenticators.

Moreover, whenever the k-th new dataset Δ_k is queried, $\mathcal{S}$ does the following. If $k = \mu$, it samples a random $\xi_\mu \in \mathbb{Z}_p$, sets $Z_\mu = (g_2^z)^{\xi_\mu}$ and stores ξ_μ. If $k \neq \mu$, it samples a random $\xi_k \in \mathbb{Z}_p$ and sets $Z_k = (g_2^v)^{\xi_k}$ and stores ξ_k. Since all Z_k are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query (Δ, τ, m, r) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k} = \mathsf{Sign}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_k \mid\mid Z_k)$.

If $k \neq \mu$ it samples $\rho_\tau \in \mathbb{Z}_p$ uniformly at random and computes $U_\tau = g_1^{-b_\tau} \cdot (g_1^u)^{\rho_\tau}$, $\Lambda_\tau = \left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j m[j]}\right)^{\frac{1}{\xi_k}}$ and gives $A = (\sigma_{\Delta_k}, Z_k, U_\tau, \Lambda_\tau)$ to $\mathcal{A}$. We have

$$e(\Lambda_\tau, Z_k) = e\left(\left((g_1^{\frac{z}{v}})^{s_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j m[j]}\right)^{\frac{1}{\xi_k}}, Z_k\right)$$

$$= e\left(\left((g_1^z)^{s_\tau} \cdot (g_1^u)^{\rho_\tau} \cdot (g_1^z)^{a_0 r + \sum_{j=1}^T a_j m[j]}\right)^{\frac{1}{v\xi_k}}, Z_k\right)$$

$$= e\left(\left((g_1^{\frac{z}{v}})^{s_\tau} \cdot g_1^{b_\tau} \cdot g_1^{-b_\tau} \cdot (g_1^{\frac{u}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^{a_0 r + \sum_{j=1}^T a_j m[j]}\right)^{\frac{1}{z_k}}, Z_k\right)$$

$$= h_\tau \cdot e(U_\tau, g_2) \cdot g_t^{a_0 z r} \prod_{j=1}^T g_t^{a_j z m[j]} = h_\tau \cdot e(U_\tau, g_2) \cdot e(\mathsf{PublicCommit}(m, r), Y).$$

This output is indistinguishable from the challenger's output during Game 7.

If $k = \mu$, simulator $\mathcal{S}$ sets $U_\tau = g_1^{-b_\tau}$, $\Lambda_\tau = (g_1^{s_\tau} \cdot g_1^{a_0 r + \sum_{j=1}^T a_j m[j]})^{\frac{1}{\xi_\mu}}$ and gives $A = (\sigma_{\Delta_\mu}, Z_\mu, U_\tau, \Lambda_\tau)$ to $\mathcal{A}$. We have

$$e\left(\Lambda_\tau, Z_\mu\right) = e\left(\left(g_1^{s_\tau} \cdot g_1^{a_0 r + \sum_{j=1}^{T} a_j m[j]}\right)^{\frac{1}{\xi_\mu}}, Z_\mu\right)$$
$$= e\left(\left(g_1^{zs_\tau + za_0 r + \sum_{j=1}^{T} za_j m[j]}\right)^{\frac{1}{z\xi_\mu}}, Z_\mu\right)$$
$$= e\left(\left(g_1^{zs_\tau} \cdot g_1^{-b_\tau} \cdot g_1^{b_\tau} \cdot g_1^{za_0 r + \sum_{j=1}^{T} za_j m[j]}\right)^{\frac{1}{z\mu}}, Z_\mu\right)$$
$$= e\left(g_1^{zs_\tau} \cdot g_1^{-b_\tau} \cdot g_1^{b_\tau} \cdot g_1^{za_0 r + \sum_{j=1}^{T} za_j m[j]}, g_2\right)$$
$$= g_t^{zs_\tau} \cdot g_t^{-b_\tau} \cdot g_t^{b_\tau} \cdot g_t^{za_0 r + \sum_{j=1}^{T} za_j m[j]} = h_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(g_1^{a_0 r + \sum_{j=1}^{T} a_j m[j]}, g_2^z\right)$$
$$= h_\tau \cdot e\left(U_\tau, g_2\right) \cdot e\left(\mathsf{PublicCommit}(m, r), Y\right)$$

and this output is indistinguishable from the challenger's output during Game 7.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, m^*, r^*, A^*)$ with $A^* = (\sigma^*_{\Delta^*}, Z^*, U^*, \Lambda^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{U}, \hat{\Lambda}, \hat{m}, \hat{r}$. If Game 7 outputs 1, we have

$$e\left(\Lambda^*, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(U^*, g_2\right) \cdot e\left(C^*, Y\right),$$

as well as

$$e\left(\hat{\Lambda}, Z_\mu\right) = \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*_{\Delta^*}) \cdot e\left(\hat{U}, g_2\right) \cdot e\left(\hat{C}, Y\right).$$

Dividing those equations yields

$$\frac{\Lambda^*}{\hat{\Lambda}} = \left(\frac{U^*}{\hat{U}} \cdot \left(\frac{C^*}{\hat{C}}\right)^z\right)^{\frac{1}{z\xi_\mu}} = \left(\frac{U^*}{\hat{U}}\right)^{\frac{1}{z\xi_\mu}} \cdot \left(\frac{C^*}{\hat{C}}\right)^{\frac{1}{\xi_\mu}}.$$

Thus $\mathcal{S}$ can compute $W = \frac{U^*}{\hat{U}}$, $W' = \left(\frac{\Lambda^*}{\hat{\Lambda}}\right)^{\xi_\mu} \cdot \frac{C^*}{\hat{C}}$. We have $(W')^z = \left(\frac{\Lambda^*}{\hat{\Lambda}}\right)^{z\xi_\mu} \cdot \left(\frac{C^*}{\hat{C}}\right)^z = \frac{U^*}{\hat{U}} = W$ and thus (W, W') is a solution to the FDHI problem. Our simulation has the same distribution as a real execution of Game 7.

□

4.3 Verifiable Computing on Shared Data from our FDC

Our instantiation of a FDC can be used to build a verifiable computing scheme for shared data supporting linear functions. Using the scheme 4.13 itself allows for information-theoretic input privacy with respect to the verifierand information-theoretic output privacy with respect to the verifier. By combining this with secret

sharing we additionally obtain information-theoretic input and output privacy with respect to the servers. That way we have shown how to achieve efficient verification and complete information-theoretic privacy.

Secure multi-party computation performed on shared data is realized using a secret sharing scheme, e.g. Shamir secret sharing [97], which we briefly describe. To share a secret $m \in \mathbb{Z}_p$, the client chooses random $a_1, \ldots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$ and computes the polynomial $P(x) = m + a_1 x + \ldots + a_{t-1}x^{t-1}$. By evaluating $P(j)$ for $j \in [N]$ it creates N shares which are given to N shareholders. Since a polynomial of degree $t-1$ is uniquely determined by t points $(j, P(j))$ one can recover the secret by requesting t shares. At the same time, even a computationally unbounded adversary cannot learn anything about m from $t-1$ shares or less (see [97]). Shamir secret sharing is linearly homomorphic, i.e. $\alpha P(j) + \beta P'(j) = (\alpha P + \beta P')(j)$ for any two polynomials $P, P' \in \mathbb{F}_p[x]$ and constants $\alpha, \beta \in \mathbb{Z}_p$. Linear functions can thus be evaluated locally on the shares.

Verifiable computing for shared data can be performed as follows.

Construction 4.27.

$\mathsf{VKeyGen}(1^\lambda, \mathcal{P})$: *On input a security parameter λ and the description of a function f given as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, it runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}', \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, *and* $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. *It sets* $\mathsf{sk} = (\mathsf{sk}', \mathcal{P})$, $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, F)$ *and returns* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$.

$\mathsf{ProbGen}(\mathsf{sk}, x)$**:** *On input the secret key* sk *and* $x = (m_1, \ldots, m_n, \Delta)$ *consisting of a tuple of n messages $m_i \in \mathbb{Z}_p^T$ for $i \in [n]$ and a dataset identifier $\Delta \in \{0,1\}^*$, it chooses $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It applies Shamir secret sharing to each message entry $m_i[j]$ for all $i \in [n], j \in [T]$ as well as to all r_i for $i \in [n]$, i.e. it computes $(s_1(m_i[j]), \ldots, s_N(m_i[j])) \leftarrow \mathsf{SShare}(m_i[j])$ for all $i \in [n], j \in [T]$, as well as $(s_1(r_i), \ldots, s_N(m_i)) \leftarrow \mathsf{SShare}(r_i)$ for all $i \in [n]$. Then, it runs $A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau)$. It chooses $k^* \in [N]$. This will identify the distinguished shareholder that will perform operations on authenticators. k^* can be chosen according to a clients preferences. It outputs the shares $s_k(m_i[j])$ as well as $s_k(r_i)$ giving $s_k(m_i[j])$ and $s_k(r_i)$ to shareholder k for $i \in [n], j \in [T], k \in [N]$, and additionally outputs $A_1, \ldots, A_n$ to shareholder k^*. It sets $\rho_x = 0$ and $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in[N]})$. It outputs (σ_x, ρ_x).*

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *On input an evaluation key* ek *and an encoded input σ_x, the algorithm parses* $\mathsf{ek} = (f, \tau_1, \ldots, \tau_n)$ *with f a linear function given by its coefficient vector $f_1, \ldots, f_n$) and* $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in[N]})$. *Each shareholder k computes $s_k(m^*[j]) = \sum_{i=1}^n f_i \cdot s_k(m_i[j])$, as well as $s_k(r^*) = \sum_{i=1}^n f_i \cdot s_k(r_i)$. They set $s_k(m^*) = (s_k(m^*[1]), \ldots, s_k(m^*[T]))$.*
Additionally k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$.

It sets $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$ *and outputs* σ_y.

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: *On input a verification key* vk*, a decoding value* ρ_x *and an encoded value* σ_y*, it parses* $\mathsf{vk} = (\mathsf{pk}, F)$*,* $\rho_x = 0$*, as well as* $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$*. It chooses a subset* $\mathcal{B} \subset [N]$ *with* $|\mathcal{B}| = t$*. It creates the reconstruction vector* $(w_1, \ldots, w_t)$ *derived from* $\mathcal{B}$ *(see Sec. 2.4 for details). It computes* $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$ *as well as* $r^* = \sum_{k\in\mathcal{B}} w_k s_k(r^*)$*. Then, it runs* $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$*.*
Finally, it runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta)$*. If* $b = 0$ *it outputs* $\perp$*, else it outputs* m^**.*

We now look at the basic properties of this construction. A first and obvious requirement is correctness, showing that any honest execution of the algorithm leads to verifiers accepting a correct result.

Proposition 4.28. *Construction 4.27 is a correct verifiable computing scheme ins the sense of Def. 2.30*

Proof. Let f be an arbitrary linear function, $x = (m_1, \ldots, m_n, \Delta)$ be an arbitrary input. Let f be described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Let $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$, $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$, and $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$.

Let $y = \sum_{i=1}^n f_i m_i$. We parse $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n], j\in[T], k\in[N]})$. Then we have

$$\begin{aligned}
C^* &= \prod_{k\in\mathcal{B}} s_k(C^*)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(s_k(y), s_k(r^*))\right)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(s_k(\sum_{i=1}^n f_i m_i), s_k(\sum_{i=1}^n f_i r_i))\right)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(\sum_{i=1}^n s_k(f_i m_i), \sum_{i=1}^n s_k(f_i r_i))\right)^{w_k}.
\end{aligned}$$

By the correctness of our FDC scheme (see Theorem 4.14) we have therefore $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) = 1$.

□

Next we consider the case of third party verifiers and show that this construction is even publicly verifiable.

Proposition 4.29. *Construction 4.27 is a publicly verifiable computing scheme.*

Proof. Note, that $\rho_x = 0$ by definition. Obviously this does not need to be kept secret. Since we have $\mathsf{vk} = (\mathsf{pk}, F)$, where pk is the public key of the FDC scheme (see Construction 4.13) and F is a function commitment. Both values are public. □

Now we formally show that this combination of our FDC with Shamir secret sharing does indeed lead to secure verifiable computing scheme.

Proposition 4.30. *Construction 4.27 is an adaptively secure verifiable computing scheme in the sense of Def. 2.33.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda]$ (see Def. 2.33), we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to either win the security experiment $\mathbf{EXP}_{\mathcal{A},\mathrm{FDC}}^{UF-CMA}$ (see Def. 4.9) or the security experiment $\mathbf{EXP}_{\mathcal{A},\mathrm{Com}}^{Bind}(\lambda)$ (see Def. 4.2).

Setup $\mathcal{S}$ runs $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$ and outputs pp. It chooses an arbitrary linear function f described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$.
Let be an , $x = (m_1, \ldots, m_m, \Delta)$ be an arbitrary input. Let f be Let $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$.

Key Generation Simulator $\mathcal{S}$ runs $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. By construction, $\mathsf{ek} = 0$. Furthermore it runs $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. It sets $\mathsf{sk} = (\mathsf{sk}', \mathcal{P})$, $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, F)$ and returns $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk})$.

Queries When $\mathcal{A}$ queries $x = (m_1, \ldots, m_n, \Delta)$ $\mathcal{S}$ does the following. It chooses $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random, and queries for $(\Delta, \tau_i, m_i, r_i)$ for $i \in [n]$, receiving A_i. It runs $\{s_k(m_i[j])\}_{k\in[N]} \leftarrow \mathsf{SShare}(m_i[j])$ for $i \in n, j \in [T]$ as well as $\{s_k(r_i\}_{k\in[N]} \leftarrow \mathsf{SShare}(r_i)$ and outputs
$\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in[N]})$. Note that this is the identical response to an honest evaluation of $\mathsf{ProbGen}$.

Forgery $\mathcal{A}$ returns σ_y^*. $\mathcal{S}$ parses $\sigma_y^* = \Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$. It chooses a subset $\mathcal{B}$ of size t and runs $m^* \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(m^*)\}_{k\in\mathcal{B}})$, as well as $r^* \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(r^*)\}_{k\in\mathcal{B}})$.
It computes $\hat{m} = \sum_{i=1}^n f_i m_i$ as well as $\hat{r} = \sum_{i=1}^n f_i r_i$. Then, it sets $\hat{C} \leftarrow \mathsf{PublicCommit}(\hat{m}, \hat{r})$ and $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$. It checks, whether $C^* = \hat{C}$. If not $(m^*, r^*), (\hat{m}, \hat{r})$ wins the binding experiment $\mathbf{EXP}_{\mathcal{A},\mathrm{Com}}^{Bind}(\lambda)$ (see Def. 4.2). If $C^* \neq \hat{C}$, it sets $\mathcal{P}_{\Delta^*}^* = (\mathcal{P}, \Delta^*)$ and outputs $(\mathcal{P}_{\Delta^*}^*, A^*, C^*)$. If $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) = 1$, then $(\mathcal{P}_{\Delta^*}^*, A^*, C^*)$ is a type 2 forgery as defined in Def. 4.8.

If we have $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda] = 1$, then in particular we have $\sigma_y^* = (\Delta,\ A^*,\ \{s_k(C^*)\}_{k\in[N]})$ such that for $C^* = \prod_{k\in\mathcal{B}} s_k(C^*)^{w_k}$ we have $\mathsf{FunctionVerify}(\mathsf{pk},\ A^*,\ C^*,\ F,\ \Delta) = 1$, which implies a forgery in $\mathbf{EXP}_{\mathcal{A},\mathrm{FDC}}^{UF-CMA}(\lambda)$.
Therefore the claim follows from Theorems 4.15 and 4.19. □

Furthermore we can show that our construction preserves efficient verification. After a one time function-dependent preprocessing verification can indeed be faster than a computation of the function itself.

Proposition 4.31. *Construction 4.27 is a verifiable computing scheme that achieves amortized efficiency in the sense of Def. 2.35.*

Proof. Let $t(n)$ be the running time of an evaluation of the linear function f. Then we have $t(n) = \mathcal{O}(n)$.

An evaluation of $\mathsf{FunctionVerify}$ consists of a signature verification, two pairing evaluations, and two group operations. Thus their combined running time is independent of n. Therefore, our construction achieves amortized efficiency for suitably large n. □

Finally we show that our verifiable computing scheme achieves complete information theoretic privacy. Over the following four propositions we prove that it offers information-theoretic input privacy with respect to the servers, information-theoretic output privacy with respect to the servers, information-theoretic input privacy with respect to the verifier and information-theoretic output privacy with respect to the verifier which have each been defined in Section 3.1.

Proposition 4.32. *Construction 4.27 achieves information-theoretic input privacy with respect to the servers in the sense of Def. 3.1 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:** Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

Let $m_1, \ldots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages. Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$. The adversary $\mathcal{A}$ chooses a subset $\mathcal{B} \subset [N]$ of size $|\mathcal{B}| = t-1$. We assume $k^* \in \mathcal{B}$. If the adversary does not corrupt k^* the claim immediately follows from the hiding property of Shamir secret sharing [97].

Thus the adversary obtains and seeks to distinguish
$(x_0, x_1, \Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$
and
$(x_0, x_1, \Delta, \{A'_i, s_k(m'_i[j]), s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
By the hiding property of Shamir secret sharing
$(x_0, x_1, \Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$
is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A_i, R_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
Obviously this is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A_i, R'_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$,
as this is just another sampling of randomness. By the hiding property of the FDC (see Theorem 4.16), this is perfectly indistinguishable from $(x_0, x_1, \Delta, \{A'_i, R'_{ijk} \mid A'_i \leftarrow$

$\mathsf{PrivateCommit}(\mathsf{sk}, m'_i, r'_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$. Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from $(x_0,\ x_1,\ \Delta,\ \{A'_i,\ s_k(m'_i[j]),\ s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$. This completes the proof.

□

Proposition 4.33. *Construction 4.27 achieves information-theoretic output privacy with respect to the servers in the sense of Def. 3.2 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:** This setup is identical to Proposition 4.32.

Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y_0 = \sum_{i=1}^n f_i m_i$, $y_1 = \sum_{i=1}^n f_i m'_i$, $r = \sum_{i=1}^n f_i r_i$, $r' = \sum_{i=1}^n f_i r'_i$.
We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y_0), s_k(r)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y_1), s_k(r')\}_{k\in[N]})$. Thus the adversary obtains and seeks to distinguish $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ and $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$.

By the hiding property of Shamir secret sharing $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R_k, S_k \mid R_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Obviously this is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R'_k, S'_k \mid R'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$ as this is just another sampling of randomness. By the hiding property of the FDC (see Theorem 4.16), this is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A', R'_k, S'_k \mid R'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$. This completes the proof.

□

Proposition 4.34. *Construction 4.27 achieves information-theoretic input privacy with respect to the verifier in the sense of Def. 3.3.*

Proof. Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

Let $m_1, \ldots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages. such that $\sum_{i=1}^n f_i m_i = \sum_{i=1}^n f_i m'_i$.
Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$. Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y = \sum_{i=1}^n f_i m_i$, $r = \sum_{i=1}^n f_i r_i$, $r' = \sum_{i=1}^n f_i r'_i$. By assumption we have $y = \sum_{i=1}^n f_i m'_i$.

We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y), s_k(r)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y), s_k(r')\}_{k\in[N]})$.

Thus the adversary obtains and seeks to distinguish
$(x_0, x_1, \Delta, A, \{s_k(y), s_k(r)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(r')\}_{k\in\mathcal{B}})$.

Note that an adversary $\mathcal{A}$ that can distinguish $(x_0, x_1, \Delta, A, \{s_k(y), s_k(r)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(r')\}_{k\in\mathcal{B}})$ immediately implies an adversary $\mathcal{A}'$ that can distinguish (x_0, x_1, Δ, y, r) and $(x_0, x_1, \Delta, y, r')$. Since both r and r' are distributed

uniformly at random as linear combinations of uniformly randomly chosen values, these are perfectly indistinguishable.

□

Proposition 4.35. *Construction 4.27 achieves information-theoretic output privacy with respect to the verifier in the sense of Def. 3.4.*

Proof. Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

We first show correctness in the sense of Def. 3.4. We provide the three additional algorithms.

$\mathsf{HideCompute}(\mathsf{ek}, \sigma_x)$: The computation algorithm takes the evaluation key ek and the encoded input σ_x.
It parses $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n], j\in[T], k\in[N]})$. Each shareholder k computes $s_k(m^*[j]) = \sum_{i=1}^n f_i \cdot s_k(m_i[j])$, as well as $s_k(r^*) = \sum_{i=1}^n f_i \cdot s_k(r_i)$. They run $s_k(C^*) \leftarrow \mathsf{PublicCommit}(s_k(m^*), s_k(r^*))$.
Additionally k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$.
It sets $\tilde{\sigma_y} = (\Delta, A^*, \{s_k(C^*)\}_{k\in[N]})$ and outputs the encoded version $\tilde{\sigma_y}$.

$\mathsf{HideVerify}(\mathsf{vk}, \tilde{\sigma_y})$: On input a verification key $\mathsf{vk} = (\mathsf{pk}, F)$ and an encoded value $\tilde{\sigma_y}$, it parses $\tilde{\sigma_y} = (\Delta, A^*, \{s_k(C^*)\}_{k\in[N]})$. It chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| = t$. It creates the reconstruction vector $(w_1, \ldots, w_t)$ derived from $\mathcal{B}$ (see Sec. 2.4 for details). Then, it runs $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$. Finally, it runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta)$. If $b = 0$ it outputs $\perp$, else it sets $\hat{\sigma_y} = (\Delta, A^*, C^*)$ outputs $\hat{\sigma_y}$.

$\mathsf{Decode}(\mathsf{vk}, \rho_x, \hat{\sigma_y}, \sigma_y)$: It takes as input the verification key vk, a decoding value $\rho_x = 0$, and encoded values $\hat{\sigma_y}, \sigma_y$. It parses $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$, $\hat{\sigma_y} = (\Delta, A^*, C^*)$. It computes $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$ and $r^* = \sum_{k\in\mathcal{B}} w_k s_k(r^*)$, and runs $b \leftarrow \mathsf{PublicDecommit}(C^*, m^*, r^*)$. If $b = 0$ it outputs $\perp$, else it returns m^*.

Now we show privacy in the sense of Def. 3.4.

Let $m_1, \ldots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages. Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$. Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y_0 = \sum_{i=1}^n f_i m_i$, $y_1 = \sum_{i=1}^n f_i m'_i$. We parse $\sigma_{y_0} = (\Delta, A, \{s_k(C)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(C')\}_{k\in[N]})$.

Note that an adversary $\mathcal{A}$ that can distinguish $(y_0, y_1, \Delta, A, \{s_k(C)\}_{k\in[N]})$ and $(y_0, y_1, \Delta, A', \{s_k(C')\}_{k\in[N]})$ immediately implies an adversary $\mathcal{A}'$ that can distinguish (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$.

However, (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$ are perfectly indistinguishable by the hiding property of the FDC (see Theorem 4.16).

□

5 Context Hiding Homomorphic Authenticators

Outsourcing costly computations to potentially untrustworthy servers leads to the risk of receiving incorrect results. Homomorphic authenticator schemes can be used to make such computations verifiable, allowing verifiers to detect incorrect results. In the case of third party verifiers this leads to a potential attack on privacy. A curious verifier can try to derive information about the inputs from the authenticator to the output. To protect against this we require schemes to provide input privacy with respect to the verifier.

Depending on the scenario, specific properties are required. To use verifiable delegation for instance for second order statistics, like variances or covariances, we require homomorphic authenticators that support quadratic functions. Before our work, schemes either did not provide input privacy with respect to the verifier or were based on multilinear maps. Multilinear maps have suffered from strong cryptanalytic attacks, and are therefore hard to instantiate.

Many interesting computations involve multiple clients; for instance, computing on health data across datasets provided by multiple hospitals. Keeping clients separate instead of merging all data supports fine-grained authenticity. In the multi-client case, a new type of attacker on the input privacy has to be considered. Where we previously considered an external verifier trying to learn about the inputs, we now also consider the scenario of one (or several) of the contributing clients learning about the inputs of a different client. Before this work, no multi-client scheme was known to provide any form of input privacy with respect to the verifier.

In this chapter, we present novel homomorphic authenticator schemes. First, to address the case of verifying quadratic functions, we present the first scheme from bilinear maps supporting quadratic functions to provide information-theoretic input privacy with respect to the verifier. Then, we present the first multi-key homomorphic authenticator scheme that achieves even information-theoretic input privacy with respect to the verifier against both the classical (external) adversaries and our newly defined (internal) adversaries.

In the previous chapter we introduced the concept of FDCs and presented a

first instantiation. Later in this thesis (see chapter 7) we will discuss how to turn suitable homomorphic authenticators into FDCs. In this chapter, we present such homomorphic authenticator schemes, fine-tailored towards specific applications – linear functions over data from multiple sources and multivariate polynomials of degree 2 respectively.

Contribution. In this chapter we present novel homomorphic signature schemes fine-tailored towards specific problems. First, we solve the problem of providing efficient and context hiding verification for multivariate quadratic functions. The core component of our solution is the new homomorphic signature scheme CHQS (Context Hiding Quadratic Signatures). CHQS allows to generate a signature on the function value of a multivariate quadratic polynomial from signatures on the input values without knowledge of the signing key. CHQS is perfectly context hiding, i.e. the signature of the output value does not leak any information about the input values. Furthermore, verification time is linear (in an amortized sense). A trade-off of our approach is a signature size that grows during homomorphic evaluation, so our scheme is not succinct. Still, freshly generated signatures are of constant size. Like most solutions in this area, the CHQS construction is based on bilinear groups. However, CHQS showcases for the first time how to use such groups to simultaneously achieve both public verification and multiplicative depth.

Next, we present the first publicly verifiable homomorphic authenticator scheme providing efficient and context hiding verification in the setting of multiple clients (allowing for multiple keys). We construct a multi-key linearly homomorphic signature scheme, and thus focus on the public key setting. We first define the context hiding property in the multi-key case. We then describe a new publicly verifiable multi-key linearly homomorphic authenticator scheme. Our scheme allows to generate an authenticator on the function value of a linear function from authenticators on the input values of various identities without knowledge of the authentication key. Furthermore, our scheme is perfectly context hiding, i.e. the authenticator to the output value does not leak any information about the input values. Using our multi-key homomorphic authenticator scheme, the verification procedure for outsourced computations of linear functions can be implemented as follows. The various clients each upload data, signed under their personal private key, to the cloud. The cloud server computes the result of the given function over these data. It also generates an authenticator to this result from the signatures on the inputs. The verifier uses this authenticator to check for correctness of the computation, by using the verification keys associated to the clients providing input to the computation. Regarding performance, verification time depends only on the number of identities involved (in an amortized sense).

Organization. This chapter is structured as follows. We first provide our definitions for the context hiding property in the multi-key scenario in Sec. 5.1. Next, we construct a multi-key homomorphic authenticator scheme that achieves information-theoretic input privacy with respect to the verifier (Sec. 5.2). After presenting the construction in Sec.5.2.1, we show its correctness and analyze its efficiency in Sec.5.2.2. We then discuss its security, first with respect to input privacy in the form of the context hiding property in Sec.5.2.3 and afterward with respect to unforgeability in Sec.5.2.4. Afterwards, we first present CHQS, our homomorphic authenticator scheme supporting quadratic functions, providing information-theoretic input privacy with respect to the verifier (Sec. 5.3). After presenting the construction in Sec. 5.3.1, we show its correctness and analyze its efficiency in Sec. 5.3.2. We then discuss its security, first with respect to input privacy in the form of the context hiding property in Sec. 5.3.3 and afterward with respect to unforgeability in Sec.5.3.4.

Publications. This chapter is based on publications [S3] and [S4].

Related Work. There are several constructions for homomorphic authenticators that go beyond the linear case. Backes et al. presented a homomorphic MAC for arithmetic circuits of degree 2 constructed from bilinear maps [12]. However, this approach is not context hiding and only offers private verifiability, while our scheme CHQS (see Construction 5.17) offers verifiability for arbitrary third parties and is perfectly context hiding. Catalano et al. showed how to construct homomorphic signatures for arithmetic circuits of fixed depth from graded encoding schemes, a special type of multilinear maps [41]. Existing graded encoding schemes [49, 61] have, however, suffered strong cryptanalytic attacks in recent years [77], [48, 82]. In contrast, CHQS can be instantiated with elliptic curve-based bilinear groups, which have long been a reliable building block in cryptography.

Some lattice-based homomorphic signatures schemes [59, 70] support boolean circuits of fixed degree. However, these schemes suffer the performance drawback of signing every single input bit, while our solution can sign entire finite field elements. Additionally [59] is also not shown to be context hiding.

There are also known multi-key homomorphic authenticators. Agrawal et al. [4] considered a notion of multi source signatures for network coding, and proposed a solution for linear functions. Network coding signatures are one application of homomorphic signatures, where signed data is combined to produce new signed data. Their solution allows for the usage of different keys in combining signatures, but differ slightly in their syntax and homomorphic property, as formalized in our definition of evaluation correctness. Unlike this work, our scheme achieves efficient verification and is perfectly context hiding. Fiore et al. [59] have even

constructed multi-key homomorphic authenticators for boolean circuits of bounded depth. While our scheme only supports linear functions, it allows the authentication of field elements, while in the case of [59] each single bit is signed individually. Thus our authenticators are significantly smaller. Both their and our solution achieve fast amortized verification, independently of function complexity. Their solution, however, is not context hiding. Lai et al. [76] proposed a generic constructions of multi-key homomorphic authenticators from zk-SNARGs. So far, zk-SNARGs are only known to exist under non-falsifiable assumptions [67]. Their constructions only allows for an a priori set bound of applications of Eval on authenticators that have been produced by Eval. Our construction has no such bound.

Verifiable computation has also been considered in the multi-key setting [46, 71]. Here the verifier is always one of the clients providing inputs to the functions, whereas our construction is publicly verifiable. Existing multi-client verifiable computation schemes also require a message from the verifier to the server, where it has to provide an encoding of the function f, which is not necessary for our homomorphic authenticators. Furthermore, the communication between the server and the verifier is at least linear in the total number of inputs of f, whereas in the case of succinct multi-key homomorphic authenticators the communication between server and verifier is proportional only to the number of clients. Finally, in multi-client verifiable computation, an encoding of one input can only be used in a single computation. Any input to be used in multiple computations has to be uploaded for each computation. In contrast, multi-key homomorphic authenticators allow the one-time authentication of every input and allow it to be used in an unbounded number of computations.

5.1 Context Hiding Multi-Key Homomorphic Authenticators

We are now ready to provide our notion of input privacy, in the form of the context hiding property. We adapt this to the multi-key setting.

Our definition for the context hiding property is inspired by Gorbunov et al.'s definition [70] for the single-key case. However, in our case, the simulator is explicitly given the circuit for which the authenticator is supposed to verify. With respect to this difference, our definition is more general. We stress that the circuit is not hidden in either of these notions. Furthermore, we differentiate between an external adversary and an internal adversary, that corrupts some of the various identities involved in a computation, i.e. knows their secret keys and inputs to a computation. Such an adversary will learn more than the outcome of the computation, since it knows some of the secret keys. It is however desirable for any non-corrupted

party to achieve context hiding privacy even against other parties involved in the computation, as far as that is possible. We now formally define context hiding for both kinds of adversaries. Finally we describe an even stronger variation of the context hiding property, where all secret keys, inputs and signatures are known. Note, that this variation is analogous to the context hiding definition of Catalano et al. [37].

Definition 5.1 (External Context Hiding)**.** *A (multi-key) homomorphic authenticator scheme for multi-labeled programs is* externally context hiding *if there exist two additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$ *and* $\mathsf{HideVer}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ *such that:*

Correctness: *For any* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{sk_{id}}, \mathsf{ek_{id}}, \mathsf{vk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and any tuple* $(\mathcal{P}_\Delta, m, \sigma)$*, such that* $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma) = 1$*, and* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, m, \sigma)$*, it holds that* $\mathsf{HideVer}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, \mathcal{P}_\Delta, m, \tilde{\sigma}) = 1$*.*

Unforgeability: *The homomorphic authenticator scheme is unforgeable in the sense of Def. 2.11 when replacing the algorithm* Ver *with* HideVer *in the security experiment.*

Context Hiding Security: *There is a simulator* Sim *such that, for any fixed (worst-case) choice of* $\{(\mathsf{sk_{id}}, \mathsf{ek_{id}}, \mathsf{vk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id}\in\mathcal{P}}$*, any multi-labeled program* $\mathcal{P}_\Delta = (f, l_1, \ldots, l_n, \Delta)$*, messages* $m_1, \ldots, m_n$*, and distinguisher* $\mathcal{D}$ *there exists a function* $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathcal{D}(\mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda),$$

where $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk_{id}}_i, \Delta, l_i, m_i)$*,* $m \leftarrow f(m_1, \ldots, m_n)$*,* $\sigma \leftarrow \mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$*, and the probabilities are taken over the randomness of* Auth, Hide *and* Sim*.*

If $\epsilon(\lambda) = \mathsf{negl}(\lambda)$*, we call the multi-key homomorphic authenticator scheme* statistically externally context hiding*. If* $\epsilon(\lambda) = 0$*, we call it* perfectly externally context hiding*.*

Note that *correctness* and *unforgeability* are the same properties as defined in Def. 2.18, only adapted to the multi-key setting.

The following definition only makes sense in the multi-key setting.

Definition 5.2 (Internal Context Hiding)**.** *A multi-key homomorphic authenticator scheme for multi-labeled programs is* internally context hiding *if there exist two additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$ *and* $\mathsf{HideVer}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ *such that:*

Correctness: *This is the same property as defined in Def. 5.1.*

Unforgeability: *This is the same property as defined in Def. 5.1.*

Context Hiding Security: *There is a simulator* Sim *such that, for any fixed (worst-case) choice of* $\{(\mathsf{sk_{id}}, \mathsf{ek_{id}}, \mathsf{vk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id}\in\mathcal{P}}$*, any multi-labeled program* $\mathcal{P}_\Delta = (f, l_1, \ldots, l_n, \Delta)$*, messages* $m_1, \ldots, m_n$*, and distinguisher* $\mathcal{D}$ *there exists a function* $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ *such that*

$$|\Pr[\mathcal{D}(\mathcal{I}, \mathsf{Hide}(\mathsf{vk}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathcal{I}, \mathsf{Sim}(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda),$$

where $\mathcal{I} = (\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\tilde{\mathsf{ID}}}, \{(m_{(\tau,\mathsf{id})}, \sigma_{(\tau,\mathsf{id})})\}_{\mathsf{id}\in\tilde{\mathsf{ID}}})$*,* $\tilde{\mathsf{ID}} \subset \mathsf{ID}$ *is a set of corrupted identities,* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk_{id}}_i, \Delta, l_i, m_i)$*,* $m \leftarrow f(m_1, \ldots, m_n)$*,* $\sigma \leftarrow \mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$*, and the probabilities are taken over the randomness of* $\mathsf{Auth}, \mathsf{Hide}$ *and* Sim*.*

If $\epsilon(\lambda) = \mathsf{negl}(\lambda)$*, we call the multi-key homomorphic authenticator scheme* statistically internally context hiding*. If* $\epsilon(\lambda) = 0$*, we call it* perfectly internally context hiding*.*

These two definitions along with Def. 2.18 provide three different notions for the context hiding property. External context hiding models an attacker that tries to distinguish a simulated authenticator derived form the result of a computation from an authenticator derived by using the homomorphic property in the form of the Eval algorithm. This external attacker has all the information a public verifier might have at his disposal, including the secret keys, modeling a leaked key. The internal attacker has all the information a public verifier might have at his disposal as well as all the information someone contributing to the computation would have, i.e. its own inputs and the authenticators it provided. Def. 2.18 presents an even stronger notion as here the distinguisher knows all the inputs and authenticators provided to the computation.

5.2 A Publicly Verifiable Multi-Key Linearly Homomorphic Authenticator Scheme

In this section, we present our multi-key homomorphic signature scheme, i.e. a publicly verifiable homomorphic authenticator. It supports linear functions. We analyze it with respect to its correctness, its succinctness and efficient verifiability. Finally, we prove that our scheme is indeed perfectly context hiding. Unforgeability is dealt with in the next section.

5.2.1 Our Construction

In our construction, a homomorphically derived authenticator consists of both components associated to an identity id and global elements. In order to prevent

the elements associated to id from leaking information about the inputs provided by id, the authenticators are randomized, and the global elements are used to deal with the randomization in order to preserve the homomorphic property. Our verification procedure naturally splits into two parts, only one of which involves the actual outcome of the computation. The other part only depends on the public verification key vk and the function to be evaluated, and can thus be precomputed. This allows for amortized efficient verification, i.e. after an expensive function-dependent one-time precomputation, all subsequent verifications occur in constant time.

Notation If we have n possibly distinct messages $m_1, \ldots, m_n$, we denote by m_i the i$^{\text{th}}$ message. Since our messages are vectors, i.e. $m \in \mathbb{Z}_p^T$, we write $m[j]$ to indicate the j$^{\text{th}}$ entry of message vector m for $j \in [T]$. Therefore $m_i[j]$ denotes the j$^{\text{th}}$ entry of the i$^{\text{th}}$ message. Given a linear function f, its i^{th} coefficient is denoted by f_i, i.e. $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$. If we have n possibly distinct authenticator components, e.g. $\Lambda_1, \ldots, \Lambda_n$, we denote by Λ_i the i$^{\text{th}}$ component. A single authenticator comprises different components, corresponding to different identities. For authenticator Λ, we denote by Λ_{id} the component for identity id. We denote by $\Lambda_{\mathsf{id},i}$ the component of the i$^{\text{th}}$ authenticator corresponding to identity id. We use a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ as a building block. $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}})$ denotes a secret/public key pair for Sig. $||$ denotes concatenation.

Construction 5.3.

MKLin*:*

$\mathsf{Setup}(1^\lambda)$: *On input a security parameter λ, this algorithm chooses the parameters $k, n, T \in \mathbb{Z}$, a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$, the message space $\mathcal{M} = \mathbb{Z}_p^T$, the tag space $\mathcal{T} = [n]$, and the identity space $\mathsf{ID} = [k]$. Additionally it fixes a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$, as well as a signature scheme Sig. It chooses $H_1, \ldots H_T \xleftarrow{\$} \mathbb{G}_1$ uniformly at random. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, F, \mathsf{Sig}, \lambda)$.*

$\mathsf{KeyGen}(\mathsf{pp})$: *On input the public parameters pp, the algorithm chooses $K \xleftarrow{\$} \mathcal{K}$ uniformly at random. It runs $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It chooses $x_1, \ldots, x_n, y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It sets $h_i = g_t^{x_i}$ for all $i \in [n]$, as well as $Y = g_2^y$. It sets $\mathsf{sk} = (K, \mathsf{sk}_{\mathsf{Sig}}, x_1, \ldots x_n, y)$, $\mathsf{ek} = \varnothing$, $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_1, \ldots, h_n, Y)$ and outputs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$. Each identity performs $\mathsf{KeyGen}()$ individually, and hence obtains its own key tuple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$: *On input a secret key sk, a dataset identifier Δ, a label $l = (\mathsf{id}, \tau)$, and a message $m \in \mathbb{Z}_p^T$, the algorithm computes $z = \Phi_K(\Delta)$, sets $Z = g_2^z$ and binds this parameter to the dataset by signing it, i.e. it*

computes $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z||\Delta)$. *Then it chooses* $r, s \in \mathbb{Z}_p$ *uniformly at random and sets* $R = g_1^{r-ys}, S = g_2^{-s}$. *It parses* $l = (\mathsf{id}, \tau)$ *and computes* $A = \left(g_1^{x_l+r} \cdot \prod_{j=1}^{T} H_j^{m[j]}\right)^{\frac{1}{z}}$ *and* $C = g_1^s \cdot \prod_{j=1}^{T} H_j^{\frac{1}{y}m[j]}$. *It sets* $\Lambda = \{(\mathsf{id}, \sigma_\Delta, Z, A, C)\}$ *and outputs* $\sigma = (\Lambda, R, S)$.

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$: *On input an function* $f : \mathcal{M}^n \to \mathcal{M}$ *and a set* $\{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]}$ *of authenticators and evaluation keys (in our construction, no evaluation keys are needed, so this set contains only authenticators), the algorithm parses* $f = (f_1, \dots f_n)$ *as a coefficient vector. It parses each* σ_i *as* (Λ_i, R_i, S_i) *and sets* $R = \prod_{i=1}^{n} R_i^{f_i}$, $S = \prod_{i=1}^{n} S_i^{f_i}$. *Set* $L_{\mathsf{ID}} = \bigcup_{i=1}^{n} \{\mathsf{id}_i\}$. *For each* $\mathsf{id} \in L_{\mathsf{ID}}$ *it chooses a pair* $(\sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}})$ *uniformly at random such that a tuple* $(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A, C)$ *is contained in one of the* Λ_i. *More formally, it chooses* $(\sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}) \xleftarrow{\$} \{(\sigma, Z) \mid \exists\ A, C \mid (\mathsf{id}, \sigma_\Delta, Z, A, C) \in \bigcup_{i=1}^{n} \Lambda_i\}$. *Then it computes* $A_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i=\mathsf{id}}}^{n} A_i^{f_i}$, $C_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i=\mathsf{id}}}^{n} C_{id_i}^{f_i}$, *and sets* $\Lambda_{\mathsf{id}} = \{(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}})\}$. *Set* $\Lambda = \bigcup_{\mathsf{id}\in L_{\mathsf{ID}}} \Lambda_{\mathsf{id}}$. *It returns* $\sigma = (\Lambda, R, S)$.

$\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$, *a set of verification key* $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$, *corresponding to the identities* id *involved in the program* $\mathcal{P}$, *a message* $m \in \mathcal{M}$, *and an authenticator* σ , *the algorithm parses* $\sigma = (\Lambda, R, S)$. *For each* id *such that* $(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}}) \in \Lambda$ *it takes* $\mathsf{pk}_{sig,\mathsf{id}}$ *from* $\mathsf{vk}_{\mathsf{id}}$ *and checks whether* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ *holds, i.e. whether there is a valid signature on* $(Z_{\mathsf{id}}||\Delta)$. *If any check fails it returns '0'. Otherwise it checks whether the following equations hold:* $\prod_{\mathsf{id}\in\mathcal{P}} e(A_{\mathsf{id}}, Z_{\mathsf{id}}) = \prod_{i=1}^{n} h_{l_i}^{f_i} \cdot \prod_{\mathsf{id}\in\mathcal{P}} e(C_{\mathsf{id}}, Y_{\mathsf{id}}) \cdot e(R, g_2)$, *as well as* $e(g_1, S) \cdot \prod_{\mathsf{id}\in\mathcal{P}} e(C_{\mathsf{id}}, Y_{\mathsf{id}}) = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$. *If they do, it outputs '1', otherwise it outputs '0'.*

Our authenticators σ consist of several components, so we have $\sigma = (\Lambda, R, S)$, where Λ is a list of elements, each associated to some identity id, i.e. $\Lambda = \{(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}})\}_{\mathsf{id}\in\mathcal{P}}$. The R and S components are global. Note, that the $A_{\mathsf{id}}, C_{\mathsf{id}}$ are randomized in order for the scheme to be internally context hiding and the global components are used to preserve the homomorphic property.

Implementation

We now report on the experimental results of a Rust implementations of Construction 5.3. The measurements are based on an implementation by Rune Fiedler and Lennart Braun. As a pairing group the BLS curve [15] *BLS12-381* [27] is used.

The following measurements were executed on an Intel Core i7-4770K (Haswell) processor running at 3.50 GHz.

We present the runtimes of the individual subalgorithms of the multi-key linearly

homomorphic authenticator scheme presented in Construction 5.3. We first present the runtimes influenced by the dimension T of vectors $m \in \mathbb{Z}_p^T$ given as messages in Table 5.1. Next, we present the runtimes influenced by the number of inputs n messages in Table 5.2. Finally, we note that the preprocessed verification EffVer depends both on the number of distinct identities k, i.e. the number of distinct keys used in this multi-key scheme, and the dimension T. These runtimes are presented in Table 5.3.

Dimension	Setup	Auth
32	6133	12671
64	12249	22077
128	24448	41162
256	48979	78769
512	97487	153549

Table 5.1: Runtimes of MKLin 5.3 in μs

Inputs	KeyGen	Eval	VerPrep
256	449467	495666	448971
512	896669	918890	897654
1024	1790956	1831059	1789746
2048	3576367	3664813	3581843
4096	7147901	7328274	7161900

Table 5.2: Runtimes of MKLin 5.3 in μs

Dimension	$k = 2$	$k = 4$	$k = 8$	$k = 16$	$k = 32$
32	28174	37536	56281	93746	168609
64	37427	46808	65564	102991	177863
128	56462	65832	84561	122035	196897
256	94457	103832	122620	160053	234944
512	170469	179806	198607	236090	310868

Table 5.3: Runtimes of EffVer in MKLin 5.3 in μs

5.2.2 Correctness and Efficiency

We now analyze our scheme with respect to its correctness and efficiency. An obvious requirement for a homomorphic authenticator scheme is to be correct.

Due to the homomorphic property, there are two different types of correctness to consider (see Def. 2.5 and Def. 2.6). The former ensures, that our scheme MKLin can be used as a conventional signature scheme, by verifying it with respect to the identity program. The latter property ensures a correct homomorphic evaluation will also be verified as correct.

Proposition 5.4. *The scheme* MKLin *(Construction 5.3) satisfies authentication correctness (see Def. 2.5), if* Sig *is a correct signature scheme.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, $l = (\mathsf{id}, \tau) \in \mathsf{ID} \times \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $m \in \mathbb{F}_p$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$. We parse $\sigma = (\Lambda, R, S)$ and $\Lambda = (\mathsf{id}, \sigma_\Delta, Z, A, C)$.

By construction we have $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z||\Delta)$ and if Sig is a correct signature scheme then $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_\Delta) = 1$ holds. We have by construction

$$e(A, Z) = e\left(\left(g_1^{x_l+r} \cdot \prod_{j=1}^{T} H_j^{m[j]}\right)^{\frac{1}{z}}, g_2^z\right) = e\left(g_1^{x_l+r-ys} \cdot g_1^{ys} \cdot \prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$$
$$= g_t^{x_l+r-ys} \cdot e\left(g_1^s \cdot \prod_{j=1}^{T} H_j^{\frac{1}{y}m[j]}, g_2^y\right) = h_l \cdot e(C, Y) \cdot e(R, g_2)$$

as well as $e(g_1, S) \cdot e(C, Y) = g_t^{-s} \cdot g_t^s e\left(\prod_{j=1}^{T} H_j^m[j], g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$, and thus $\mathsf{Ver}(\mathcal{I}_{l,\Delta}, \mathsf{vk}, m, \sigma) = 1$ holds. □

Proposition 5.5. *The scheme* MKLin *(Construction 5.3) satisfies evaluation correctness (see Def 2.6).*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow Setup1^\lambda$ be arbitrary public parameters, $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id} \in \mathsf{ID}}$ be a set of arbitrary key triples, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier. Let $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i \in [N]}$ be an arbitrary set of program/message/authenticator triples, such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$. Let $g : \mathcal{M}^N \to \mathcal{M}$ be an arbitrary linear function given by its coefficient vector $(g_1, \ldots, g_N)$ Let $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, \{\sigma_i\}_{i \in [N]})$.

We will make use of the following notation: We write $\mathsf{id} \in \mathcal{P}$ if for $\mathcal{P} = (f, l_1, \ldots, l_n)$ there exists an $i \in [n]$ such that $l_i = (\mathsf{id}, \tau_i)$ for some input identifier τ_i.

Since we have $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$, we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ for all $\mathsf{id} \in \mathcal{P}_i$. Note that we have $\bigcup_{i=1}^{N} \{\mathsf{id} \in \mathcal{P}_i\} = \{\mathsf{id} \in \mathcal{P}^*\}$. Therefore we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ holds for all $\mathsf{id} \in \mathcal{P}^*$

If $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$, holds, then in particular

$$\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(A_{\mathsf{id},i}, Z_{\mathsf{id}}\right) = \prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} C_{\mathsf{id},i}, Y_{\mathsf{id}}\right) \cdot e\left(R_i, g_2\right)$$

holds as well as $e\left(g_1, S_i\right) \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} C_{\mathsf{id},i}, g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m_i[j]}, g_2\right)$ for all $i \in [N]$. Without loss of generality let $\{\mathsf{id} \in \mathcal{P}_{i,\Delta}\} = \{\mathsf{id} \in \mathcal{P}_{j,\Delta}\}$ for all $i, j \in [n]$. Let f_k for $k \in [n]$ denote the coefficients describing $\mathcal{P} = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. Then we have $f_k = \sum_{i=1}^{N} g_i f_{i,k}$. We have

$$\prod_{i=1}^{N}\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(A_{\mathsf{id},i}, Z_{\mathsf{id},i}\right)\right)^{g_i} = \prod_{i=1}^{N}\left(\prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot \prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(C_{\mathsf{id},i}, Y_{\mathsf{id}}\right) \cdot e\left(R_i, g_2\right)\right)^{g_i}$$

and

$$\begin{aligned}
&\prod_{\mathsf{id}\in\mathcal{P}^*_{\Delta}} e\left(A^*_{\mathsf{id}}, Z^*_{\mathsf{id}}\right) = \prod_{i=1}^{N}\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(A_{\mathsf{id},i}, Z_{\mathsf{id},i}\right)\right)^{g_i} \\
&= \prod_{i=1}^{N}\left(\prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot \prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(C_{\mathsf{id},i}, Y_{\mathsf{id}}\right) \cdot e\left(R_i, g_2\right)\right)^{g_i} \\
&= \prod_{k=1}^{n} h_{l_k}^{f_k} \cdot \prod_{\mathsf{id}\in\mathcal{P}^*_{\Delta}} e\left(C^*_{\mathsf{id}}, Y^*_{\mathsf{id}}\right) \cdot e\left(R^*, g_2\right)
\end{aligned}$$

We also have

$$\begin{aligned}
&e\left(g_1, S^*\right) \cdot \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right) = e\left(g_1, \prod_{i=1}^{N} S_i^{g_i}\right) \cdot \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\prod_{i=1}^{N} C_{i,\mathsf{id}}^{g_i}, Y_{\mathsf{id}}\right) \\
&= \prod_{i=1}^{N} e\left(\prod_{j=1}^{T} H_j^{m_i[j]}, g_2\right)^{g_i} = e\left(\prod_{j=1}^{T} H_j^{\sum_{i=1}^{N} g_i m_i[j]}, g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m^*[j]}, g_2\right).
\end{aligned}$$

Thus all checks of $\mathsf{Ver}()$ pass and $\mathsf{Ver}(\mathcal{P}^*_{\Delta}, \mathsf{vk}, m^*, \sigma^*) = 1$ holds. □

We now consider our scheme's efficiency properties, first w.r.t. bandwidth, in the form of succinctness, and then w.r.t. verification time.

A trivial solution to constructing a homomorphic signature scheme is to (conventionally) sign every input, and during Eval to just concatenate all the signatures along with the corresponding values. Verification then consists of checking every input value and then redoing the computation. This naive solution is obviously undesirable in terms of bandwidth, efficiency and does not provide any privacy guarantees.

Succinctness guarantees that a homomorphically derived signature is still small, thus keeping bandwidth requirements low. Efficient verification ensures that the time required to check an authenticator is low. This is achieved by splitting Ver into two sub-algorithms, one of which can be precomputed, and the other one EffVer can be faster than natively computing the function itself.

Proposition 5.6. *The scheme* MKLin *(Construction 5.3) is succinct (Def. 2.7).*

Proof. An authenticator consists of (at most) $k+1$ elements of $\mathbb{G}_1$, k elements of $\mathbb{G}_2$, k identities $\mathsf{id} \in \mathsf{ID}$, and k (conventional) signatures. None of this depends on the input size n. Therefore MKLin is succinct. □

Proposition 5.7. *The scheme* MKLin *(Construction 5.3) allows for efficient verification (Def.2.8).*

Proof. We describe the algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$:

$\mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}})$: On input a labeled program $\mathcal{P} = (f, l_1, \ldots, l_n)$, with f given by its coefficient vector $(f_1, \ldots f_n)$, the algorithm takes $(Y_{\mathsf{id}}, \mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}})$ from $\mathsf{vk}_{\mathsf{id}}$. For label $l_i = (\mathsf{id}_i, \tau_i)$ it takes h_{l_i} from $\mathsf{vk}_{\mathsf{id}_i}$. It computes $h_{\mathcal{P}} \leftarrow \prod_{i=1}^{n} h_{l_i}^{f_i}$ and outputs $\mathsf{vk}_{\mathcal{P}} \leftarrow (h_{\mathcal{P}}, \{(Y_{\mathsf{id}}, \mathsf{pk}_{sig,\mathsf{id}})\}_{\mathsf{id}\in\mathcal{P}})$. This is independent of the input size n.

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$**:** On input a concise verification key $\mathsf{vk}_{\mathcal{P}}$, a dataset Δ, a message m, and an authenticator σ, the algorithm parses $\sigma = (\Lambda, R, S)$. For each $\mathsf{id} \in \mathcal{P}$ it checks whether $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ holds. If not, it outputs '0'. Otherwise, it checks whether the following equation holds: $\prod_{\mathsf{id}\in\mathcal{P}} e\left(A_{\mathsf{id}}, Z_{\mathsf{id}}\right) = h_{\mathcal{P}} \cdot \prod_{\mathsf{id}\in\mathcal{P}} e\left(C_{\mathsf{id}}, Y_{\mathsf{id}}\right) \cdot e\left(R, g_2\right)$ as well as $e\left(g_1, S\right) \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}} C_{\mathsf{id}}, g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$. If they do, it outputs '1', otherwise it outputs '0'.

Obviously if $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma) = b$ and $\mathsf{vk}_{\mathcal{P}} \leftarrow \mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}})$, then also $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma) = b$. We can see that the runtime of $\mathsf{EffVer}()$ is $\mathcal{O}(k)$, and is independent of the input size n. Thus, for $n \gg k$, MKLin allows for efficient verification. □

5.2.3 Context Hiding

We now showcase our scheme's privacy property. On a high level, we want an authenticator to the output of a computation not to leak information about the inputs to the computation, which we have formalized in Definitions. 5.1 and 5.2. Intuitively, the outcome of a function (e.g. the average) reveals significantly less information than the individual inputs to the computation. In our scenario, multiple clients upload data to a cloud server that performs the computation, and allows for public verification of the result due to the use of homomorphic authenticators.

The context hiding property ensures that the verifier cannot use the authenticators provided to him to derive additional information about the inputs, beyond his knowledge of the output.

We recall that in the multi-key setting, the question who exactly is meant to be prevented from learning about the input values becomes relevant. In particular, we differentiate between an external adversary — one that has not corrupted any of the identities involved in a computation — and an internal adversary, who has this additional knowledge. Thus we capture the two slightly different notions of keeping the input values private with respect to some outside party (*externally context hiding*), versus keeping the input values confidential with respect to an computation (*internally context hiding*). The second property is naturally stronger. We show that our scheme from Construction 5.3 achieves even the stronger property.

Theorem 5.8. *The scheme* MKLin *(Construction 5.3) is perfectly internally context hiding (Def. 5.2) and thus also externally context hiding (Def. 5.1).*

Proof. First, in our case, the algorithm Hide is just the identity function. More precisely, we have $\mathsf{Hide}(\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{ID}}, m, \sigma) = \sigma$, for all possible verification keys $\mathsf{vk}_{\mathsf{id}}$, messages m and authenticators σ. Thus we have $\mathsf{HideVer} = \mathsf{Ver}$, so correctness and unforgeability hold by Proposition 5.4, Proposition 5.5, and Theorem 5.9.

We show how to construct a simulator Sim that outputs authenticators perfectly indistinguishable from the ones obtained by running Eval. Consider that for all linear functions f, we have $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i = \sum_{i\in\mathcal{I}} f_i m_i + \sum_{j\in\mathcal{J}} f_j m_j$, for each $\mathcal{I}, \mathcal{J} \subset [n]$, with $\mathcal{I} \cup \mathcal{J} = [n]$ and $\mathcal{I} \cap \mathcal{J} = \varnothing$.

$\mathcal{S}$ can simulate the corrupted parties perfectly. By the identity shown before, we can in our case therefore reduce internal context hiding security to external context hiding security. We now show external context hiding security. Parse the simulator's input as $\mathsf{sk}_{\mathsf{id}} = (K_{\mathsf{id}}, K'_{\mathsf{id}}, \mathsf{sk}_{\mathsf{Sig}_{\mathsf{id}}}, x_{1,\mathsf{id}}, \ldots x_{n,\mathsf{id}}, y_{\mathsf{id}})$, $m = (m[1], \ldots, m[T])$, and $\mathcal{P}_\Delta = (f, l_1, \ldots, l_n, \Delta)$. With this information, the simulator computes:

$Z'_{\mathsf{id}} = g_2^{z_{\mathsf{id}}}$ where $z_{\mathsf{id}} \leftarrow \Phi_{K_{\mathsf{id}}}(\Delta)$	$\sigma'_{\Delta,\mathsf{id}} \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}\|\|\Delta)$
$r'_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$	$r' = \sum_{l=(\tau,\mathsf{id})\in\mathcal{P}} f_l r'_{\mathsf{id}}$
$s'_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$	$s' = \sum_{\mathsf{id}\in\mathcal{P}} y_{\mathsf{id}} s_l$
$A'_{\mathsf{id}} = \left(g_1^{\sum_{(\mathsf{id},\tau)\in\mathcal{P}} x_l f_l + r'} \cdot \prod_{j=1}^{T} H_j^{m[j]}\right)^{\frac{1}{z_{\mathsf{id}}}}$	$\mathsf{id}^* \xleftarrow{\$} \mathsf{ID}$
$C'_{\mathsf{id}} = g_1^{-s'_{\mathsf{id}}}$ for all $\mathsf{id} \neq \mathsf{id}^*$	$C'_{\mathsf{id}^*} = g_1^{-s'_{\mathsf{id}^*}} \cdot \prod_{j=1}^{T} H_j^{\frac{1}{y} m[j]}$
$R' = g_1^{r' + \sum_{\mathsf{id}\in\mathcal{P}} y_{\mathsf{id}} s'_{\mathsf{id}}}$	$S' = g_2^{-s'}$
$\Lambda' = \bigcup_{\mathsf{id}\in\mathcal{P}} \{(A'_{\mathsf{id}}, Z'_{\mathsf{id}}, \sigma'_{\Delta,\mathsf{id}})\}$	

The simulator outputs the authenticator $\sigma' = (\Lambda', R', S')$. We now show that this simulator allows for perfectly context hiding security. We fix arbitrary key

pairs $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$, a multi-labeled program $\mathcal{P}_\Delta$, and messages $m_1, \ldots, m_n \in \mathbb{Z}_p^T$.

Let $\sigma \leftarrow \mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$ and parse it as $\sigma = (\Lambda, R, S)$. We look at each component of the authenticator. We have $Z_{\mathsf{id}} = \Phi_{K_{\mathsf{id}}}(\Delta)$ by definition and therefore also $Z_{\mathsf{id}} = Z'_{\mathsf{id}}$. Y_{id} and Y'_{id} are both taken from the public keys and therefore identical. In particular we also have $z_{\mathsf{id}} = z'_{\mathsf{id}}$. We have $\sigma_{\Delta_{\mathsf{id}}} = \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}', Z_{\mathsf{id}}||\Delta)$ by definition and $\sigma'_{\Delta_{\mathsf{id}}} = \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}', Z'_{\mathsf{id}}||\Delta)$. Since $Z_{\mathsf{id}} = Z'_{\mathsf{id}}$, for all $\mathsf{id} \in \mathcal{P}$, $\sigma_{\Delta_{\mathsf{id}}}$ and $\sigma'_{\Delta_{\mathsf{id}}}$ are identically distributed and thus perfectly indistinguishable to any distinguisher $\mathcal{D}$. These components are therefore either identical or perfectly indistinguishable to any distinguisher $\mathcal{D}$.

A'_{id} is a uniformly random (u.r.) element of $\mathbb{G}_1$, as r'_{id} is also u.r. A_{id} is a u.r. element of $\mathbb{G}_1$, as $r_{\mathsf{id}} = \sum_{\mathsf{id}\in l} f_l r_l$ is u.r. as a linear combination of u.r. elements. C'_{id} is a u.r. element of $\mathbb{G}_1$, as s'_{id} is also u.r. C_{id} is a u.r. element of $\mathbb{G}_1$, as $s_{\mathsf{id}} = \sum_{\mathsf{id}\in l} f_l s_l$ is u.r. as a linear combination of u.r. elements. $R'$is a u.r. element of $\mathbb{G}_1$, as all r'_{id} are also u.r. R is a u.r. element of $\mathbb{G}_1$, as $r = \sum_{\mathsf{id}\in\mathcal{P}} r_{\mathsf{id}}$ is u.r. as a linear combination of u.r. elements. The $A'_{\mathsf{id}}, C'_{\mathsf{id}}$ as well as R' uniquely define S' and $A_{\mathsf{id}}, C_{\mathsf{id}}$ as well as R uniquely define S.

Thus, all simulated elements have the identical distribution as the ones from the real evaluation. They correspond to a different choice of randomness during Auth. This holds even if all secret keys $\mathsf{sk}_{\mathsf{id}}$ are known to $\mathcal{D}$. Hence σ and σ' are perfectly indistinguishable for any (computationally unbounded) distinguisher $\mathcal{D}$. □

5.2.4 Unforgeability

In delegated computations, the question of the correctness of the result arises. Homomorphic authenticators aim at making these computations verifiable, thus allowing for the detection of incorrect results. It should therefore be infeasible for any adversary to produce a authenticator that passes the $\mathsf{Ver}()$ algorithm, that has not been produced by honestly performing the $\mathsf{Eval}()$ algorithm. This has been formalized in Def 2.11. In this section, we present the security reduction for the unforgeability of our scheme. To this end, we first describe a sequence of games, allowing us to argue about different variants of forgeries. We then present a series of lemmata, where we bound the difference between those games.

Since our authenticators have multiple components, we consider specific types of forgeries in the various games, i.e. ones where one or multiple components are indeed correct, and in our final security reduction we consider the generic case. When simulating the final two games, the issue of providing signatures, without knowing the correct secret key arises. Here we use the elements h_{id,τ_i} taken from the public keys associated to the label $l = (\mathsf{id}, \tau_i)$ and embed an information theoretically hidden trapdoor into them, which we use to answer signing queries. Note, that by (conventionally) signing the concatenation $(\Delta||Z_{\mathsf{id}})$ we use a similar approach to Fiore et al. [59, Theorem 2]. Not directly using their more generic

approach however yields a lower bound on the adversary's overall success probability in the security experiment.

Theorem 5.9. *The scheme* MKLin *(Construction 5.3) is unforgeable in the sense of Def. 2.11 if* Sig *is an unforgeable (EU-CMA [69]) signature scheme,* Φ *is a pseudorandom function and* $\mathcal{G}$ *is a bilinear group generator, such that the DL assumption (see Def. 2.36), the DDH assumption (see Def. 2.38) and the FDHI assumption (see Def. 2.42) hold.*

Proof. We can deal with corruptions via our generic result of Proposition 2.15. It is thus sufficient to prove the security against adversaries that make no corruptions. Recall that any corrupted party provides their key tuples $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ to the adversary, giving the adversary additional knowledge in order for him to adaptively query messages. To prove Theorem 5.9, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. any game outputs '1', only with negligible probability. Following the notation of [37], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We use flag values bad_i, initially set to false. If, at the end of each game, any of these previously defined flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during game i. Using Proposition 2.16, any adversary who outputs a Type 3 forgery (see Def. 2.10) can be converted into one that outputs a Type 2 forgery. Hence we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where $\mathcal{A}$ makes no corruption queries and only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 1, except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 5.10, we show that Bad_2 cannot occur if Sig is unforgeable.

Game 3 is defined as Game 2, except that the keyed pseudorandom function F_K is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. In Lemma 5.11, we show that these two games are indistinguishable if F is pseudorandom.

Game 4 is defined as Game 3, except for the following changes. It computes $\hat{m} = f^*(m_1, \ldots, m_n)$, as well as $\hat{\sigma} = \mathsf{Eval}(f^*, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$, i.e. it runs an honest computation over the queried messages and generated authenticators in dataset Δ^*. The challenger runs an additional check. If $\prod_{j=1}^{T} H_j^{m^*[j]} = \prod_{j=1}^{T} H_j^{\hat{m}[j]}$ and $\hat{m} \neq m^*$ it sets $\mathsf{bad}_4 = \mathsf{true}$. We clearly have $|\Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_4]$. In Lemma 5.12, we show that any adversary $\mathcal{A}$ for which Bad_4 occurs implies a solver for the DL problem.

Game 5 is defined as Game 4, except for the following change. The challenger runs an additional check. If $\prod_{\mathsf{id}\in\mathcal{P}} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right) = \prod_{\mathsf{id}\in\mathcal{P}} e\left(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ and $m^* \neq \hat{m}$

it sets $\mathsf{bad}_5 = \mathsf{true}$, where C^*_{id} are components of the forged authenticator σ^* and $\hat{C}_{\mathsf{id}}$ are components of the honest execution of Eval over the queried data set, as defined in Game 4. We have $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_5]$. In Lemma 5.14, we show that any adversary $\mathcal{A}$ for which Bad_5 occurs implies a solver for the DDH problem.

Game 6 is defined as Game 5, except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_6 = \mathsf{true}$. In Lemma 5.13, we show that $\Pr[G_5(\mathcal{A})] \leq Q \cdot \Pr[G_6(\mathcal{A})]$.

Game 7 is defined as Game 6, except for the following change. The challenger runs an additional check. If $\mathsf{Ver}(\mathcal{P}^*_{\Delta^*}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, m^*, \sigma^*) = 1$ as well as $\hat{m} \neq m^*$ and $\prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z^*_{\mathsf{id}}\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z^*_{\mathsf{id}}\right)$, where $\hat{A}_{\mathsf{id}}, A^*_{\mathsf{id}}$ are the components taken from $\hat{\sigma}$ and σ^* respectively, then $\mathcal{C}$ sets $\mathsf{bad}_7 = \mathsf{true}$. We have $|\Pr[G_6(\mathcal{A})] - \Pr[G_7(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_7]$. In Lemma 5.15, we show that any adversary $\mathcal{A}$ for which Bad_7 occurs implies a solver for the FDHI problem.

Finally, Lemma 5.16 shows that any adversary $\mathcal{A}$ that wins Game 7 implies a solver for the FDHI problem. Together, Lemma 5.10—5.16 prove Theorem 5.9 and we have $\Pr[\mathcal{G}(\mathcal{A})] \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda) + \mathsf{Adv}^{PRF}_{F,\mathsf{D}}(\lambda) + (1 - \frac{1}{p}) \cdot \mathsf{Adv}^{DL}_{\mathcal{S}}(\lambda) + \mathsf{Adv}^{DDH}_{\mathcal{S}}(\lambda) + 2Q\mathsf{Adv}^{FDHI}_{\mathcal{S}}(\lambda)$. □

Lemma 5.10. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_2(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.20. □

Lemma 5.11. *For every PPT adversary $\mathcal{A}$ running Game 3, there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{\Phi,\mathsf{D}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.21.

□

Lemma 5.12. *For every PPT adversary $\mathcal{A}$ running Game 4, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_4] \leq \mathsf{Adv}^{DL}_{\mathcal{S}}(\lambda)$.

Proof. This is a direct corollary of Theorem 4.15. □

Lemma 5.13. *For every PPT adversary $\mathcal{A}$ running Game 6, we have* $\Pr[G_5(\mathcal{A})] \leq Q \cdot \Pr[G_6(\mathcal{A})]$.

Proof. This is a direct corollary of Lemma 4.22. □

Lemma 5.14. *For every PPT adversary $\mathcal{A}$ running Game 5, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_5] \leq \mathsf{Adv}^{DDH}_{\mathcal{S}}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 5. We will show how a simulator $\mathcal{S}$ can use this to break the DDH problem in $\mathbb{G}_1$. It takes as input a tuple $(\mathsf{bgp}, g_1^x, g_1^y, g_1^z)$.

Setup Simulator $\mathcal{S}$ sets $\mathsf{bgp}' = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1^x, g_2, e)$ and chooses $c_j \in \mathbb{F}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$.

Queries Simulator $\mathcal{S}$ can run KeyGen honestly and answer any authentication queries honestly.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 5 to compute $\hat{m}, \hat{\sigma}$. We then have

$$e(g_1^x, S^*) \cdot \prod_{\mathsf{id} \in \mathcal{P}} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right) = e(\prod_{j=1}^{T} H_j^{m^*[j]}, g_2) = e(g_1^{\sum_{j=1}^{T} c_j m^*[j]}, g_2)$$

as well as

$$e(g_1^x, \hat{S}) \cdot \prod_{\mathsf{id} \in \mathcal{P}} e\left(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}}\right) = e(\prod_{j=1}^{T} H_j^{\hat{m}[j]}, g_2) = e(g_1^{\sum_{j=1}^{T} c_j \hat{m}[j]}, g_2).$$

We set $M = \sum_{j=1}^{T} c_j(m^*[j] - \hat{m}[j])$. Dividing the two equations and using the fact that $\prod_{\mathsf{id} \in \mathcal{P}} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right) = \prod_{\mathsf{id} \in \mathcal{P}} e\left(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ we obtain

$$e\left(g_1^x, \frac{S^*}{\hat{S}}\right) = e\left(g_1, g_2\right)^M.$$

We now have $z = xy$ if and only if $e\left(g_1^y, g_2\right) = e\left(g_1^z, \left(\frac{S^*}{\hat{S}}\right)^{\frac{1}{M}}\right)$. Since $\mathsf{bad}_4 = \mathsf{false}$, we have $M \neq 0$.
Note that since we have $\mathsf{bad}_4 = \mathsf{false}$ we must have $S^* \neq \hat{S}$.

□

Lemma 5.15. *For every PPT adversary $\mathcal{A}$ running Game 7, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_7] = \mathsf{Adv}_{\mathcal{S}}^{FDHI}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 7 such that $\mathsf{bad}_7 = \mathsf{true}$. We will show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ simulator $\mathcal{S}$ simulates Game 7.

Setup Simulator $\mathcal{S}$ chooses $c_j \in \mathbb{Z}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, \mathcal{R}, \mathsf{Sig}, \lambda)$.

Key Generation Simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. During the key generation it chooses $a_l, b_l \in \mathbb{Z}_p$ uniformly at random for all $l \in \mathcal{L}$. It sets $h_l = g_t^{a_l} \cdot e\left(g_1, g_2^z\right)^{b_l}$. It honestly runs $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$, chooses y_{id} uniformly at random, and sets $Y_{\mathsf{id}} = (g_2^z)^{y_{\mathsf{id}}}$ for all $\mathsf{id} \in \mathsf{ID}$. It gives the public keys $\mathsf{ek}_{\mathsf{id}} = \varnothing$, $\mathsf{vk}_{\mathsf{id}} = (\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, h_{\mathsf{id},1}, \ldots, h_{\mathsf{id},n}, Y_{\mathsf{id}})$ to $\mathcal{A}$ for all $\mathsf{id} \in \mathsf{ID}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$. (Initially it sets $k = 1$). For every new queried dataset Δ simulator $\mathcal{S}$ creates a list L_Δ of pairs (l, m), which collects all the label/message pairs queried by the adversary on Δ and the respectively generated authenticators.
Moreover, whenever the k^{th} new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_{\mathsf{id},\mu} \in \mathbb{Z}_p$, for all $\mathsf{id} \in \mathsf{ID}$ sets $Z_{\mathsf{id},\mu} = (g_2^z)^{\xi_{\mathsf{id},\mu}}$ and stores $\xi_{\mathsf{id},\mu}$. If $k \neq \mu$, it samples a random $\xi_{\mathsf{id},k} \in \mathbb{Z}_p$ and sets $Z_{\mathsf{id},k} = (g_2^v)^{\xi_{\mathsf{id},k}}$ and stores $\xi_{\mathsf{id},k}$. Since all $Z_{\mathsf{id},k}$ are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query (Δ, l, m) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k,\mathsf{id}} = \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}_{\mathsf{id}}}, (\Delta_k || Z_{k,\mathsf{id}}))$.
If $k \neq \mu$, it samples $\rho_l, s_l \in \mathbb{Z}_p$ uniformly at random and computes

$$
\begin{aligned}
A_l &= \left((g_1^{\frac{z}{v}})^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot (g_1^{\frac{r}{v}})^{\rho_l} \right)^{\frac{1}{\xi_{\mathsf{id},k}}}, \\
R_l &= g_1^{-a_l} \cdot (g_1^r)^{\rho_l}, \\
S_l &= g_2^{-s_l}, \textbf{as well as} \\
C_l &= g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}
\end{aligned}
$$

It sets $\Lambda_l = (\mathsf{id}, \sigma_{\Delta_k,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$
\begin{aligned}
e\left(A_l, Z_{k,\mathsf{id}}\right) &= e\left(\left(\left(g_1^{\frac{z}{v}} \right)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot \left(g_1^{\frac{r}{v}} \right)^{\rho_l} \right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^v)^{\xi_{\mathsf{id},k}} \right) \\
&= e\left((g_1^z)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot (g_1^r)^{\rho_l}, g_2 \right) \\
&= e\left(g_1^{b_l z + a_l - a_l + r\rho_l}, g_2 \right) \cdot e\left(g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}, g_2^{z y_{\mathsf{id}}} \right) \\
&= h_l \cdot e\left(R, g_2\right) \cdot e\left(C, Y_{\mathsf{id}}\right)
\end{aligned}
$$

as well as

$$
e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s + s + \sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2 \right)
$$

and this output is indistinguishable from the challenger's output during Game 7.

If $k = \mu$, simulator $\mathcal{S}$ computes $A_l = \left((g_1^{\frac{z}{v}})^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \right)^{\frac{1}{\xi_{\mathsf{id},k}}}$, $R_l = g_1^{-a_l}$, $C_l = g_1^{s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, as well as $S_l = g_2^{-s_l}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{\Delta_k,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$\begin{aligned} e\left(A_l, Z_{k,\mathsf{id}}\right) &= e\left(\left(g_1^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^z)^{\xi_{\mathsf{id},k}}\right) \\ &= e\left((g_1^z)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}, g_2\right) \\ &= e\left(g_1^{b_l z + a_l - a_l}, g_2\right) \cdot e\left(g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}, g_2^{z y_{\mathsf{id}}}\right) = h_l \cdot e\left(R, g_2\right) \cdot e\left(C, Y_{\mathsf{id}}\right) \end{aligned}$$

and

$$e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s + s + \sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$$

so this output is indistinguishable from the challenger's output during Game 7.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{m}, \hat{\sigma}$. Since $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is a successful forgery, we have

$$\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(R^*, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$$

as well as

$$\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(\hat{R}, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}}\right)$$

according to Proposition 5.5. We compute $A^* = \prod_{\mathsf{id} \in \mathcal{P}^*} (A^*_{\mathsf{id}})^{\xi_{\mathsf{id},\mu}}$ as well as $\hat{A} = \prod_{\mathsf{id} \in \mathcal{P}^*} \left(\hat{A}_{\mathsf{id}}\right)^{\xi_{\mathsf{id},\mu}}$. We note that $e\left(A^*, g_2^z\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right)$ and $e\left(\hat{A}, g_2^z\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right)$. Since we have $\mathsf{bad}_7 = \mathsf{true}$, we know that $A^* = \hat{A}$. We compute $C^* = \prod_{\mathsf{id} \in \mathcal{P}^*} (C^*_{\mathsf{id}})^{y_{\mathsf{id}}}$ as well as $\hat{C} = \prod_{\mathsf{id} \in \mathcal{P}^*} \left(\hat{C}_{\mathsf{id}}\right)^{y_{\mathsf{id}}}$. We have $e\left(C^*, g_2\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ and $e\left(\hat{C}, g_2\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}})$.
By dividing the equations above and using $A^* = \hat{A}$, we obtain $e\left(\frac{\hat{C}}{C^*}, g_2^z\right) = e\left(\frac{R^*}{\hat{R}}, g_2\right)$. Setting $W = \frac{\hat{C}}{C^*}$ and $W' = \frac{R^*}{\hat{R}}$, we get a solution (W, W') to the FDHI assumption. Since $\mathsf{bad}_5 = \mathsf{false}$, we know that $C^* \neq \hat{C}$ and thus $(W, W') \neq (1, 1)$.

□

Lemma 5.16. *For every PPT adversary $\mathcal{A}$ running Game 7, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[G_7(\mathcal{A})] = \mathsf{Adv}_{\mathcal{S}}^{FDHI}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 7. We will show how a simulator $\mathcal{S}$ can use this to break the FDHI problem. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ simulator $\mathcal{S}$ simulates Game 7.

Setup Simulator $\mathcal{S}$ chooses $c_j \in \mathbb{Z}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, \mathcal{R}, \mathsf{Sig}, \lambda)$.

Key Generation Simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. During the key generation, it chooses $a_l, b_l \in \mathbb{Z}_p$ uniformly at random for all $l \in \mathcal{L}$. It sets $h_l = g_t^{a_l} \cdot e\left(g_1, g_2^z\right)^{b_l}$. It honestly runs $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$, chooses y_{id} uniformly at random, and sets $Y_{\mathsf{id}} = g_2^{y_{\mathsf{id}}}$ for all $\mathsf{id} \in \mathsf{ID}$. Note, that unlike Lemma 5.15, we do not use the element g_2^z taken from the problem instance to generate the Y_{id}. It gives the public keys $\mathsf{ek}_{\mathsf{id}} = \varnothing$, $\mathsf{vk}_{\mathsf{id}} = (\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, h_{\mathsf{id},1}, \ldots, h_{\mathsf{id},n}, Y_{\mathsf{id}})$ to $\mathcal{A}$ for all $\mathsf{id} \in \mathsf{ID}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ simulator $\mathcal{S}$ creates a list L_Δ of pairs (l, m), which collects all the label/message pairs queried by the adversary on Δ and the respectively generated authenticators.
Moreover, whenever the k^{th} new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_{\mathsf{id},\mu} \in \mathbb{Z}_p$, for all $\mathsf{id} \in \mathsf{ID}$ sets $Z_{\mathsf{id},\mu} = (g_2^z)^{\xi_{\mathsf{id},\mu}}$ and stores $\xi_{\mathsf{id},\mu}$. If $k \neq \mu$, it samples a random $\xi_{\mathsf{id},k} \in \mathbb{Z}_p$ and sets $Z_{\mathsf{id},k} = (g_2^v)^{\xi_{\mathsf{id},k}}$ and stores $\xi_{\mathsf{id},k}$. Since all $Z_{\mathsf{id},k}$ are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query (Δ, l, m) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k,\mathsf{id}} = \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}_{\mathsf{id}}}, (\Delta_k || Z_{k,\mathsf{id}}))$.
If $k \neq \mu$, it samples $\rho_l, s_l \in \mathbb{Z}_p$ uniformly at random and computes $A_l = \left((g_1^{\frac{z}{v}})^{b_l} \cdot (g_1^{\frac{r}{v}})^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}$, $R_l = g_1^{-a_l - y_{\mathsf{id}} s_l} \cdot (g_1^r)^{\rho_l} \cdot g_1^{-\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, $S_l = g_2^{-s_l}$, as well as $C_l = g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{\Delta_k,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.

We have

$$
\begin{aligned}
e\left(A_l, Z_{k,\mathsf{id}}\right) &= e\left(\left((g_1^{\frac{z}{v}})^{b_l} \cdot (g_1^{\frac{r}{v}})^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^{v})^{\xi_{\mathsf{id},k}}\right) = e\left(g_1^{zb_l + r\rho_l}, g_2\right) \\
&= e\left(g_1^{zb_l + a_l - a_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j] - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j] s y_{\mathsf{id}} s_l - y_{\mathsf{id}} s_l + r\rho_l}, g_2\right) \\
&= g_t^{a_l + zb_l} \cdot e\left(g_1^{-a_l} \cdot (g_1^{r})^{\rho_l} \cdot g_1^{-y_{\mathsf{id}} s_l - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}, g_2^{Y}\right) \\
&= h_l \cdot e\left(R_l, g_2\right) \cdot e\left(C_l, Y_{\mathsf{id}}\right)
\end{aligned}
$$

and

$$
e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s + s + \sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right).
$$

This output is indistinguishable from the challenger's output during Game 7.
If $k = \mu$, simulator $\mathcal{S}$ computes $A_l = \left(g_1^{b_l}\right)^{\frac{1}{\xi_{\mathsf{id},\mu}}}$, $C_l = g_1^{s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, $R_l = g_1^{-y_{\mathsf{id}} s_l - a_l - \sum_{j=1}^{T} c_j m[j]}$, as well as $S_l = g_2^{-s_l}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{\Delta_k,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$
\begin{aligned}
e\left(A_l, Z_{\mu,\mathsf{id}}\right) &= e\left(\left(g_1^{b_l}\right)^{\frac{1}{\xi_{\mathsf{id},\mu}}}, (g_2^{z})^{\xi_{\mathsf{id},\mu}}\right) = e\left(g_1^{zb_l}, g_2\right) \\
&= e\left(g_1^{zb_l + a_l - a_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j] - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j] + y_{\mathsf{id}} s_l - y_{\mathsf{id}} s_l}, g_2\right) \\
&= g_t^{a_l + zb_l} \cdot e\left(g_1^{-y_{\mathsf{id}} s_l - a_l - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot g_1^{y_l s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}, g_2\right) \\
&= h_l \cdot e\left(R_l, g_2\right) \cdot e\left(C_{\mathsf{id}}, Y_{\mathsf{id}}\right)
\end{aligned}
$$

and

$$
e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s + s + \sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right).
$$

This output is indistinguishable from the challenger's output during Game 7.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{m}, \hat{\sigma}$. Since $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is a successful forgery, we have

$$
\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(R^*, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)
$$

as well as

$$\prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(\hat{R}, g_2\right) \cdot \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}}\right)$$

according to Proposition 5.5. We compute $A^* = \prod_{\mathsf{id}\in\mathcal{P}^*} (A^*_{\mathsf{id}})^{\xi_{\mathsf{id},\mu}}$ as well as $\hat{A} = \prod_{\mathsf{id}\in\mathcal{P}^*} \left(\hat{A}_{\mathsf{id}}\right)^{\xi_{\mathsf{id},\mu}}$. We note that $e\left(A^*, g_2^z\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right)$ and $e\left(\hat{A}, g_2^z\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right)$. Since we have $\mathsf{bad}_7 = \mathsf{false}$ we know that $A^* \neq \hat{A}$. We compute $C^* = \prod_{\mathsf{id}\in\mathcal{P}^*} (C^*_{\mathsf{id}})^{y_{\mathsf{id}}}$ as well as $\hat{C} = \prod_{\mathsf{id}\in\mathcal{P}^*} \left(\hat{C}_{\mathsf{id}}\right)^{y_{\mathsf{id}}}$. We note that $e\left(C^*, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ and $e\left(\hat{C}, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}})$. Dividing the equations above, we obtain $e\left(\frac{A^*}{\hat{A}}, g_2^z\right) = e\left(\frac{C^*\cdot R^*}{\hat{C}\cdot\hat{R}}, g_2\right)$ and setting $W = \frac{R^*\cdot C^*}{\hat{R}\cdot\hat{C}}$, as well as $W' = \frac{A^*}{\hat{A}}$ we have obtained a solution (W, W') to the FDHI assumption. Since we have $\mathsf{bad}_7 = \mathsf{false}$, we know that $A^* \neq \hat{A}$ and thus $(W, W') \neq (1, 1)$.

□

5.3 A Context Hiding Homomorphic Signature Scheme for Quadratic Functions

5.3.1 CHQS: A New Homomorphic Signature Scheme for Quadratic Functions

We now present the algorithms making up CHQS. It is homomorphic with respect to arithmetic circuits $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ of degree 2, where $p \geq 5$ (see Prop. 2.17). CHQS is *graded*, i.e. there exist level-1 and level-2 signatures. Level-1 signatures are created by signing messages, whereas level-2 signatures occur during homomorphic evaluation over multiplication gates. Graded structures like this occur naturally in homomorphic schemes like the ones by Catalano and others [34, 41]. We use dedicated elements (which we will denote by T_τ) in our level-1 signatures to handle multiplication gates. Those elements no longer occur in the level-2 signatures.

Construction 5.17.

$\mathsf{Setup}(1^\lambda)$*: On input a security parameter λ the algorithm runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. It chooses $n \in \mathbb{N}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, \mathsf{bgp}, \mathsf{Sig}, \Phi)$.*

$\mathsf{KeyGen}(\mathsf{pp})$ ***:*** *On input public parameters* pp *it chooses* $x, y \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random. It sets* $h_t = g_t^x$*. It samples* $t_{\tau_i}, k_{\tau_i} \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random for all* $i \in [n]$ *and sets* $F_{\tau_i} = g_2^{t_{\tau_i}}$*, as well as* $f_{\tau_i} = g_t^{y t_{\tau_i}}$*,* $f_{\tau_i,\tau_j} = g_t^{t_{\tau_i} k_{\tau_j}}$*, for all* $i, j \in [n]$*. Additionally the algorithm chooses a random seed* $K \xleftarrow{\$} \mathcal{K}$ *for the pseudorandom function* Φ*. It computes keys for the regular signature scheme* $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*. It sets* $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, K, x, y, \{t_{\tau_i}\}_{i=1}^n, \{k_{\tau_i}\}_{i=1}^n)$*,* $\mathsf{ek} = 0$ *and* $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_t, \{F_{\tau_i}, f_{\tau_i}\}_{i=1}^n, \{f_{\tau_i,\tau_j}\}_{i,j=1}^n)$*.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$***:*** *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* $\tau \in \mathcal{T}$*, and a message* $m \in \mathbb{Z}_p$*, the algorithm generates the parameters for the dataset identified by* Δ*, by running* $z \leftarrow \Phi_K(\Delta)$ *and computing* $Z = g_2^{\frac{1}{z}}$*. Z is bound to the dataset identifier* Δ *by using the regular signature scheme, i.e. it sets* $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta || Z)$*.*

It chooses $r, s \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random. It computes* $\Lambda \leftarrow g_1^{z(xm+(y+s)t_\tau+r)}$*,* $R \leftarrow g_1^r$*,* $S_\tau \leftarrow g_1^s$*, as well as* $T_\tau \leftarrow g_1^{ym-k_\tau}$*. It sets* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ *and then returns the signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$*. Following the convention of Backes et al. [12], our signature contains the message* m*.*

$\mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$***:*** *Inputs are a public evaluation key* ek*, an arithmetic circuit* f *of degree at most 2, and signatures* $\sigma_1, \ldots, \sigma_n$*, where (w.l.o.g.)* $\sigma_i = (m_i, \sigma_{\Delta,i}, Z_i, \Lambda_i, R_i, \mathcal{T}_i)$*. The algorithm checks if the signatures share the same public values, i.e. if* $\sigma_{\Delta,1} = \sigma_{\Delta,i}$ *and* $Z_1 = Z_i$ *for all* $i = 2, \ldots, n$*, and the signature for each set of public values is correct and matches the dataset identifier* Δ*, i.e.* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta_i || Z_i) = 1$ *for any* $i \in [n]$*. If this is not the case, the algorithm rejects the signature. Otherwise, it proceeds as follows. We describe this algorithm in terms of six different procedures* $(\mathsf{Add}_1, \mathsf{Mult}, \mathsf{Add}_2, \mathsf{cMult}_1, \mathsf{cMult}_2, \mathsf{Shift})$ *allowing to evaluate the circuit gate by gate.*

- Add_1***:*** *On input two level-1 signatures* $\sigma_i = (m_i, \sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{T}_i)$ *for* $i = 1, 2$ *it computes as follows:* $m = m_1 + m_2$*,* $\Lambda = \Lambda_1 \cdot \Lambda_2$*,* $R = R_1 \cdot R_2$*, and* $S_\tau = S_{\tau,1} \cdot S_{\tau,2}$ *as well as* $T_\tau = T_{\tau,1} \cdot T_{\tau,2}$ *for all* $(\tau, \cdot) \in \mathcal{T}_1 \cap \mathcal{T}_2$*,* $S_\tau = S_{\tau,i}$ *as well as* $T_\tau = T_{\tau,i}$ *for all* τ *such that* $(\tau, \cdot) \in \mathcal{T}_1 \Delta \mathcal{T}_2$*, and* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ *for all* $(\tau, \cdot) \in \mathcal{T}_1 \cup \mathcal{T}_2$*. It outputs a level-1 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$*.*
- Mult***:*** *On input two level-1 signatures* $\sigma_i = (m_i, \sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{T}_i)$ *for* $i = 1, 2$ *and the public key* pk*, it computes as follows:* $m = m_1 m_2$*,* $\Lambda = \Lambda_1^{m_2}$*,* $R = R_1^{m_2}$*,* $S'_{\tau_1} = S_{\tau_1}^{m_2} \cdot \prod_{\tau_2 \in \mathcal{T}_2} T_{\tau_2}$*, for all* $\tau_1 \in \mathcal{T}_1$*, and* $\mathcal{L} = \{(\tau, S'_\tau)\}$ *for all* $\tau \in \mathcal{T}_1$*. It outputs a level-2 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*.*
- Add_2***:*** *On input two level-2 signatures* $\sigma_i = (m_i, \sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{L}_i)$ *for* $i = 1, 2$*, it computes as follows:* $m = m_1 + m_2$*,* $\Lambda = \Lambda_1 \cdot \Lambda_2$*,* $R = R_1 \cdot R_2$*,* $S_\tau = S_{\tau,1} \cdot S_{\tau,2}$ *for all* $(\tau, \cdot) \in \mathcal{L}_1 \cap \mathcal{L}_2$*,* $S_\tau = S_{\tau,i}$ *for all* τ *such that* $(\tau, \cdot) \in \mathcal{L}_1 \Delta \mathcal{L}_2$*, and* $\mathcal{L} = \{(\tau, S_\tau)\}$ *for all* $(\tau, \cdot) \in \mathcal{L}_1 \cup \mathcal{L}_2$*. It outputs a*

level-2 signature $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$.

cMult_1*: On input a level-1 signature* $\sigma' = (m', \sigma_\Delta, Z, \Lambda', R', \mathcal{T}')$ *and a constant* $c \in \mathbb{Z}_p$*, it computes as follows:* $m = cm'$, $\Lambda = \Lambda'^c$, $R = R'^c$, $S_\tau = S'^c_\tau$, $T_\tau = T'^c_\tau$ *for all* $\tau \in \mathcal{T}'$*, and* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}_{\tau \in \mathcal{T}}$*. It outputs a level-1 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$.

cMult_2*: On input a level-2 signature* $\sigma = (m', \sigma_\Delta, Z, \Lambda', R', \mathcal{L}')$ *and a constant* $c \in \mathbb{Z}_p$*, it computes as follows:* $m = cm'$, $\Lambda = \Lambda'^c$, $R = R'^c$, $S_\tau = S'^c_\tau$ *for all* $(\tau, S'_\tau) \in \mathcal{L}'$*, and* $\mathcal{L} = \{(\tau, S_\tau)\}$ *for all* $(\tau, S'_\tau) \in \mathcal{L}'$*. It outputs a level-2 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$.

Shift*: On input a level-1 signature* $\sigma' = (m', \sigma_\Delta, Z, \Lambda', R', \mathcal{T}')$*, it computes as follows:* $m = m'$, $\Lambda = \Lambda'$, $R = R'$, *and* $\mathcal{L} = \{(\tau, S_\tau)\}_{\tau \in \mathcal{T}'}$*. It outputs a level-2 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$. Shift *simply describes how to derive a level-2 signature from a level-1 signature.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, M, \sigma)$*: On input a public evaluation key* vk*, a message* M*, a (level-1 or -2) signature* σ*, a multi-labeled program* $\mathcal{P}_\Delta$ *containing an arithmetic circuit* f *of degree at most 2, the algorithm parses (without loss of generality)* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$.
It then checks whether the following conditions hold:

1. $M = m$
2. $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta || Z) = 1$
3. $$e(\Lambda, Z) = e(R, g_2) \cdot h_t^m \cdot \prod_{i=1}^{n} f_i^{c_i} \cdot \prod_{(\tau,\cdot,\cdot) \in \mathcal{T}} e(S_\tau, F_\tau)$$
   as well as for an arbitrary $\tau^* \in \mathcal{T}$
   $$e\left(\prod_{\tau \in \mathcal{T}} T_\tau, F_{\tau^*}\right) \cdot \prod_{\tau \in \mathcal{T}} f_{\tau,\tau^*}^{c_\tau} = f_{\tau^*}^m$$
   for level-1 signatures and
   $$e(\Lambda, Z) = e(R, g_2) \cdot h_t^m \cdot \prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^{n} f_j^{c_j} \cdot \prod_{(\tau,\cdot) \in \mathcal{L}} e(S_\tau, F_\tau)$$
   for level-2 signatures, respectively, where $c_{i,j}$ *and* c_j *are the coefficients in* $\mathcal{P}_\Delta$.

If all 4 *or* 3 *conditions hold respectively, it returns '1'. Otherwise, it returns '0'.*

Implementation

We now report on the experimental results of a Rust implementations of Construction 5.17. The measurements are based on an implementation by Rune Fiedler and

Lennart Braun. As a pairing group the BLS curve [15] *BLS12-381* [27] is used.

The following measurements were executed on an Intel Core i7-4770K (Haswell) processor running at 3.50 GHz.

We present the runtimes of the individual subalgorithms of the authenticator scheme for quadratic functions presented in Construction 5.17. We present the runtimes influenced by the number of inputs n messages in Table 5.4. Note that the runtime of Auth is independent of n and as in this scheme messages $m \in \mathbb{Z}_p$ are given as inputs, the dimension of inputs does not vary. The result is given in Table 5.5.

Inputs	KeyGen	Eval	VerPrep	EffVer
1	8549	2920	3440	8692
2	14688	7568	8667	10985
4	32315	22107	24351	15619
8	88630	72157	76282	24823
16	285081	256934	264901	43149
32	1015105	958685	976165	79983
64	3823874	3703191	3733905	153457
128	14833281	14631758	14597335	300702
256	58396099	58071618	57725493	594869
512	231798428	232029549	229621235	1183757
1024	923505377	933255024	915832701	2361895
2048	3686773222	3795627595	3657874966	4721335

Table 5.4: Runtimes of CHQS 5.17 in μs

Auth	2138

Table 5.5: Runtimes of Authentication for CHQS 5.17 in μs

5.3.2 CHQS: Correctness and Efficiency

We will now analyze CHQS with respect to its basic correctness and efficiency properties.

The following proposition shows that freshly generated authenticators, created by calling Auth are accepted by Ver.

Proposition 5.18. CHQS *(Construction 5.17) achieves authentication correctness in the sense of Def- 2.5 if* Sig *is a correct signature scheme.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, $\tau \in \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $m \in \mathbb{F}_p$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. We parse $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$.

We obviously have $m = m$. If Sig is a correct signature scheme we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. Then we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(g_1^{z(xm+(y+s)t_\tau+r)}, g_2^{\frac{1}{z}}\right) = e\left(g_1^{xm+(y+s)t_\tau+r}, g_2\right) \\ &= g_t^{xm+(y+s)t_\tau+r} = h_t^m \cdot f_\tau \cdot e(R, g_2) \cdot e(S, F_\tau) \end{aligned}$$

Therefore all checks of Ver pass.

□

We now show the second correctness property, by showing that if we have several authenticators that are accepted by Ver and we apply Eval to them we again obtain an authenticator that is accepted by Ver.

Proposition 5.19. CHQS *(Construction 5.17) achieves evaluation correctness in the sense of Def- 2.6.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier.

We show the correctness of the evaluation of the six procedures (Add_1, Mult, Add_2, cMult_1, cMult_2, Shift). So we take any two program/message/authenticator triples $(\mathcal{P}_i, m_i, \sigma_i)$ for $i = 1, 2$, such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$.

Note, that without loss of generalization we can assume $Z_1 = Z_2$. If we had $Z_1 \neq Z_2$, and we know that since $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for $i = 1, 2$ we in particular have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. If we had $\sigma^* \leftarrow \mathsf{Eval}(\mathsf{ek}, g, \sigma_1, \sigma_2)$, $\mathcal{P}^* = g(\mathcal{P}_1, \mathcal{P}_2)$, $m^* = g(m_1, m_2)$ and $\mathsf{Ver}(\mathcal{P}^*, \mathsf{vk}, m^*, \sigma^*) = 0$, then we find σ_1', σ_2' with $Z_1 = Z_2$ such that the same holds. To achieve this we can simply set $\sigma_1' = \sigma_1$ and $\sigma_2' = (m_2, \sigma_{\Delta,1}, Z_1, R_2, \mathcal{T}_2)$ or $\sigma_2' = (m_2, \sigma_{\Delta,1}, Z_1, R_2, \mathcal{L}_2)$ depending on whether σ_2 is a level 1 or 2 signature. We then have $\mathsf{Eval}(\mathsf{ek}, g, \sigma_1, \sigma_2) = \mathsf{Eval}(\mathsf{ek}, g, \sigma_1', \sigma_2')$. Thus we can assume that $Z_1 = Z_2$.

- Add_1**:** Since we have $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for $i = 1, 2$, we know that in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $m_1 + m_2 = g(m_1, m_2)$.

Furthermore we have

$$\begin{aligned}
&e(\Lambda, Z) = e(\Lambda_1 \cdot \Lambda_2, Z_1) \\
&= e(\Lambda_1, Z_1) \cdot e(\Lambda_2, Z_2) \\
&= e(R_1, g_2) \cdot h_t^{m_1} \cdot f_1 \cdot e(S_1, F_1) \cdot e(R_2, g_2) \cdot h_t^{m_2} \cdot f_2 \cdot e(S_2, F_2) \\
&= e(R_1 \cdot R_2, g_2) \cdot h_t^{m_1+m_2} \cdot f_1 \cdot f_2 \cdot e(S_1, F_1) \cdot e(S_2, F_2) \\
&= e(R, g_2) \cdot h_t^{m} \cdot f_1 \cdot f_2 \cdot e(S_1, F_1) \cdot e(S_2, F_2)
\end{aligned}$$

as well as for an arbitrary $\tau^* \in \mathcal{T}$

$$\begin{aligned}
&e\left(\prod_{\tau\in\mathcal{T}} T_\tau, F_{\tau^*}\right) \cdot f_1 \cdot f_2 = e\left(\prod_{\tau\in\mathcal{T}_1} T_\tau \cdot \prod_{\tau\in\mathcal{T}_2} T_\tau, F_{\tau^*}\right) \cdot f_1 \cdot f_2 \\
&= e\left(\prod_{\tau\in\mathcal{T}_1} T_\tau, F_{\tau^*}\right) \cdot f_1 \cdot e\left(\prod_{\tau\in\mathcal{T}_2} T_\tau, F_{\tau^*}\right) \cdot f_2 \\
&= f_{\tau^*}^{m_1} \cdot f_{\tau^*}^{m_2} = f_{\tau^*}^{m_1+m_2}
\end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

Mult: Since we have $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for $i = 1, 2$, we know that in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. For ease of notation we consider the case where $\mathcal{T}_i$ each contains only a single entry. We also have $m_1 \cdot m_2 = g(m_1, m_2)$. Furthermore we have

$$\begin{aligned}
&e(\Lambda, Z) = e(\Lambda_1^{m_2}, Z_1) = e(\Lambda_1, Z_1)^{m_2} \\
&= e(R_1^{m_2}, g_2) \cdot h_t^{m_1 m_2} \cdot f_1^{m_2} \cdot e(S_1^{m_2}, F_1) \\
&= e(R_1^{m_2}, g_2) \cdot h_t^{m_1 m_2} \cdot f_1^{m_2} \cdot e(S_1^{m_2}, F_1) \cdot e(T_2, F_1) \cdot f_{1,2} \cdot f_1^{-m_2} \\
&= e(R, g_2) \cdot h_t^{m} \cdot f_{1,2} \cdot e(S_1^{m_2} \cdot T_2, F_1) = e(R, g_2) \cdot f_{1,2} \cdot e(S, F_1)
\end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

Add$_2$: Since we have $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for $i = 1, 2$, we know that in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $m_1 + m_2 = g(m_1, m_2)$. Furthermore we have

$$\begin{aligned}
&e(\Lambda, Z) = e(\Lambda_1^{m_2}, Z_1) = e(\Lambda_1, Z_1)^{m_2} \\
&= e(R_1, g_2) \cdot h_t^{m_1} \cdot f_1 \cdot \prod_{(\tau,\cdot)\in\mathcal{L}_1} e(S_{\tau,1}, F_\tau) \\
&\qquad \cdot e(R_2, g_2) \cdot h_t^{m_2} \cdot f_2 \cdot \prod_{(\tau,\cdot)\in\mathcal{L}_2} e(S_{\tau,2}, F_\tau) \\
&= e(R, g_2) \cdot h_t^{m} \cdot f_1 \cdot f_2 \prod_{(\tau,\cdot)\in\mathcal{L}} e(S_\tau, F_\tau)
\end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

cMult_1: Since we have $\mathsf{Ver}(\mathcal{P}'_\Delta, \mathsf{vk}, m', \sigma') = 1$, we know in particular that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$. So with $Z = Z'$ and $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $m = c \cdot m' = g(m')$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(\Lambda'^c, Z'\right) = e(\Lambda', Z')^c \\ &= e\left(R'^c, g_2\right) \cdot h_t^{cm'} \cdot f'^c \cdot e\left(S'^c, F\right) \\ &= e(R, g_2) \cdot h_t^m \cdot f \cdot e(S, F) \end{aligned}$$

as well as for an arbitrary $\tau^* \in \mathcal{T}$

$$\begin{aligned} & e\left(\prod_{\tau \in \mathcal{T}} T_\tau, F_{\tau^*}\right) \cdot fe\left(\prod_{\tau \in \mathcal{T}'} {T'}_\tau^c, F_{\tau^*}\right) \cdot f'^c \\ &= \left(e\left(\prod_{\tau \in \mathcal{T}'} {T'}_\tau, F_{\tau^*}\right) \cdot f'\right)^c \\ &= \left(f_{\tau^*}^{m'}\right)^c = f_{\tau^*}^m \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

cMult_2: Since we have $\mathsf{Ver}(\mathcal{P}'_\Delta, \mathsf{vk}, m', \sigma') = 1$, we know that in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$. So with $Z = Z'$ and $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $m = c \cdot m' = g(m')$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(\Lambda'^c, Z'\right) = e(\Lambda', Z')^c \\ &= e\left(R'^c, g_2\right) \cdot h_t^{cm'} \cdot f'^c \cdot e\left(S'^c, F\right) \\ &= e(R, g_2) \cdot h_t^m \cdot f \cdot e(S, F) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

Shift: Since we have $\mathsf{Ver}(\mathcal{P}'_\Delta, \mathsf{vk}, m', \sigma') = 1$, we know that in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$ holds. So with $Z = Z'$, $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $m = m' = g(m')$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e(\Lambda', Z') \\ &= e(R', g_2) \cdot h_t^{m'} \cdot f' \cdot e(S', F) \\ &= e(R, g_2) \cdot h_t^m \cdot f \cdot e(S, F) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathcal{P}, \mathsf{vk}, m, \sigma)$ pass.

□

We now analyze the runtime of our verification algorithm Ver.

Proposition 5.20. CHQS *(Construction 5.17) provides verification in time* $\mathcal{O}(n)$ *in an amortized sense.*

Proof. We describe the two algorithms (VerPrep, EffVer).

VerPrep($\mathsf{pk}, \mathcal{P}$) : This algorithm parses $\mathcal{P} = (f, \tau_1, \ldots \tau_n)$ with $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} c_i m_i + \sum_{i,j=1}^{n} c_{i,j} m_i m_j$ and takes the $f_i, f_{i,j}$ for $i, j \in [n]$ contained in the public key. It computes $F_{\mathcal{P}} \leftarrow \prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{i=1}^{n} f_i^{c_i}$ and outputs $\mathsf{vk}_{\mathcal{P}} = (\mathsf{pk}_{\mathsf{Sig}}, h_t, \{F_i\}_{i=1}^{n}, F_{\mathcal{P}})$ where $\mathsf{pk}_{\mathsf{Sig}}, h_t, \{F_i\}_{i=1}^{n}$ are taken from vk.

EffVer($\mathsf{vk}_{\mathcal{P}}, m, \sigma, \Delta$)**:** This algorithm is analogous to Ver, except that the value $\prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{i=1}^{n} f_i^{c_i}$ has been precomputed as $F_{\mathcal{P}}$.

This satisfies correctness. During EffVer, the verifier now computes

$$e(\Lambda, Z) = e(R, g_2) \cdot h_t^m \cdot F_{\mathcal{P}} \cdot \prod_{i=1}^{n} e(S_i, F_i)$$

The running time of EffVer is thus $\mathcal{O}(n)$. □

Thus, CHQS achieves amortized efficiency in the sense of Def. 2.8 for every arithmetic circuit f of multiplicative depth 2, that has *superlinear* runtime complexity.

Bandwidth: CHQS is *not succinct*. However, the output of Auth is of constant size and thus independent of n. Hence no extensive bandwidth is needed during the upload of the data. After a homomorphic evaluation a signature consists of up to $n + 2$ elements in $\mathbb{G}_1$, 1 element in $\mathbb{G}_2$, one conventional signature, the message and up to n input identifiers contained in a list.

5.3.3 CHQS: Context Hiding Property

Next, we consider the privacy property of CHQS. More specifically, we show that it achieves information-theoretic input privacy with respect to the verifier. For homomorphic authenticators we discussed variations of the context hiding property in Section 2.2. Since CHQS is a single key scheme there is no difference between the internal (see Def. 5.2) and external (see Def. 5.1) context hiding property.

Theorem 5.21. CHQS *(Construction 5.17) is perfectly externally context hiding according to Definition 5.1.*

Proof. We show that our scheme is perfectly externally context hiding in the sense of Def. 5.1, by comparing the distributions of homomorphically derived signatures to that of simulated signatures. In our construction we have to distinguish between level-1 and level-2 signatures. For level-2 signatures Hide is just the identity function, i.e. $\sigma \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ for all vk, m, σ and $\mathsf{HideVer} = \mathsf{Ver}$. For level-1 signatures Hide is the operation Shift described in Eval and we again have $\mathsf{HideVer} = \mathsf{Ver}$. We show how to construct a simulator Sim that outputs signatures perfectly indistinguishable from the ones obtained by running Eval. Parse the simulator's input as $\mathsf{sk} = (\mathsf{sk}', K)$, $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$. For each τ appearing in $\mathcal{P}_\Delta$, it chooses $s_\tau \in \mathbb{Z}_p$ uniformly at random as well as $r \in \mathbb{Z}_p$ uniformly at random. With this information, the simulator computes $m' = m$, $Z' = g_2^z$ where $z \leftarrow \Phi_K(\Delta)$, $\sigma'_\Delta \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}', Z||\Delta)$, $\Lambda' = g_1^{z(xm' + y(\sum_{i,j=1}^n c_{ij} t_i k_j + \sum_{i=1}^n c_i t_i) + \sum_{i=1}^n s_{\tau_i} t_i + r)}$, $R' = g_1^r$, $S'_\tau = g_1^{s_\tau}$ for all τ appearing in $\mathcal{P}_\Delta$, and $\mathcal{L}' = \{(\tau, S_\tau)\}_{\tau \in \mathcal{P}_\Delta}$. The simulator outputs the signature $\sigma' = (m', \sigma'_\Delta, Z', \Lambda', R', \mathcal{L}')$.

We now show that this simulator allows for perfectly context hiding security. We fix an arbitrary key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$, a multi-labeled program $(f, \tau_1, \ldots, \tau_n, \Delta)$, and messages $m_1, \ldots, m_n \in \mathbb{Z}_p$. Let $\sigma \leftarrow \mathsf{Eval}(\mathsf{vk}, \mathcal{P}_\Delta, \sigma_1, \ldots, \sigma_n)$ and parse it as $\sigma = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$. We inspect each component of the signature. $Z = \Phi_K(\Delta)$ by definition and thus also $Z = Z'$. In particular, $z = z'$ where $Z = g_2^z$ and $Z' = g_2^{z'}$. We have $\sigma_\Delta = \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}', Z||\Delta)$ by definition, and since $Z = Z'$, σ_Δ and σ'_Δ are identically distributed. We consider Λ as an exponentiation of g_1^z. Since $\Lambda = \prod_{i,j=1}^n \Lambda_i^{c_{ij} m_j}$ by construction, for the exponent we have:

$$xm + \sum_{i,j=1}^{n} c_{ij} m_j (s_i t_i + y t_i + r_i) = xm' + \sum_{i,j=1}^{n} c_{ij} m_j (s_i t_i + y t_i + r_i)$$
$$+ y(\sum_{i,j=1}^{n} c_{ij} t_i k_j + \sum_{i=1}^{n} c_i t_i) - y(\sum_{i,j=1}^{n} c_{ij} t_i k_j + \sum_{i=1}^{n} c_i t_i)$$

$$= xm' + \sum_{i=1}^{n} (\sum_{j=1}^{n} -y c_{ij} k_j - y c_i + c_{ij} m_j y + c_{ij} s_i m_j) t_i$$
$$+ y(\sum_{i,j=1}^{n} c_{ij} t_i k_j + \sum_{i=1}^{n} c_i t_i) + \sum_{i,j=1}^{n} c_{ij} m_j r_i$$
$$= xm' + y(\sum_{i,j=1}^{n} c_{ij} t_i k_j + \sum_{i=1}^{n} c_i t_i) + \sum_{i=1}^{n} \tilde{s}_i t_i + \tilde{r}$$

Thus the exponent corresponds to a different choice of $r, s_i \in \mathbb{Z}_p$. Analogously, $S_\tau = g_1^{\tilde{s}_{\tau_i}}$ for all $(\tau, \cdot) \in \mathcal{L}_1$ and $R = g_1^{\tilde{r}}$, where $\tilde{r}, \tilde{s}_{\tau_i}$ are distributed uniformly at random as linear combinations of uniformly random field elements.

All elements are either identical, or have the exact same distribution. Thus even a computationally unbounded distinguisher has no advantage distinguishing the two cases. □

5.3.4 CHQS: Unforgability

In this section we will discuss the unforgeability of CHQS. We provide a security reduction by describing various security games, dealing with specific types of forgeries and bound the difference between these games in a series of lemmata. Under various cryptographic assumptions all of these differences,as well as the final probability of an adversary winning the security game are negligible.

Theorem 5.22. CHQS *(Construction 5.17) is unforgeable in the sense of Def. 2.11, if* Sig *is an unforgeable (EU-CMA [69]) signature scheme,* Φ *is a pseudorandom function and* $\mathcal{G}$ *is a bilinear group generator, such that the FDHI assumption (see Def. 2.42) holds.*

Proof. To prove Theorem 5.22, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1' only with negligible probability. Following the notation of [37], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We use flag values bad_i, initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during game i. As shown in Proposition 2.17, any adversary who outputs a Type 3 forgery (see Def. 2.9) can be converted into one that outputs a Type 2 forgery. Hence we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where $\mathcal{A}$ makes no corruption queries and only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 1, except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 5.23, we show that Bad_2 cannot occur if Sig is unforgeable. It is worth noticing that after this change the game never outputs '1'if $\mathcal{A}$ returns a Type 1 forgery.

Game 3 is defined as Game 2, except that the keyed pseudorandom function Φ_K is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. In Lemma 5.24 we show that these two games are indistinguishable if Φ is pseudorandom.

Game 4 is defined as Gam 3, except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of

queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_4 = \mathsf{true}$. In Lemma 5.25 we show that $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Game 5 is defined as Game 4, except for the following change. At the very beginning $\mathcal{C}$ chooses $z_\mu \in \mathbb{Z}_p$ at random and computes $Z_\mu = g_2^{z_\mu}$. It will use Z_μ whenever queried for dataset Δ_μ. It chooses $a_i, b_i \in \mathbb{Z}_p$ uniformly at random for $i \in [n]$ and sets $f_{\tau_i} = g_t^{y(a_i + z_\mu b_i)}$, $F_{\tau_i} = g_2^{a_i + z_\mu b_i}$ as well as $f_{\tau_i,\tau_j} = g_t^{k_j y(a_i + z_\mu b_i)}$. In Lemma 5.26, we show that $\Pr[G_5(\mathcal{A})] = \Pr[G_4(\mathcal{A})]$.

Game 6 is defined as Game 5, except for the following change. The challenger runs an additional check. If $\mathsf{Ver}(\mathsf{pk}, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$, the challenger computes $\hat{\sigma} \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}^*_\Delta, \boldsymbol{\sigma})$ over the signatures σ_i generated in dataset Δ^*. We have $\hat{\sigma} = (\hat{m}, \hat{\mathcal{T}}, \sigma_\Delta,\ Z, \hat{\Lambda}, \hat{R}, \hat{S})$ in case of a level-1 signature and $\hat{\sigma} = \sigma = (\hat{m}, \sigma_\Delta, Z, \hat{\Lambda}, \hat{R}, \hat{\mathcal{L}})$ in case of a level-2 signature. If $\Lambda^* \cdot \prod_{i=1}^n \hat{S}_i^{b_i} = \hat{\Lambda} \cdot \prod_{i=1}^n S_i^{*b_i}$, then $\mathcal{C}$ sets $\mathsf{bad}_6 = \mathsf{true}$. In Lemma 5.27, we show that any adversary $\mathcal{A}$ for which Bad_6 occurs implies a solver for the FDHI problem.
Finally, in Lemma 5.28, we show that any adversary $\mathcal{A}$ that wins Game 6 also implies a solver for the FDHI problem.

□

Lemma 5.23. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_2(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.20.

□

Lemma 5.24. *For every PPT adversary $\mathcal{A}$ running Game 3, there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{\Phi,\mathsf{D}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.21.

□

Lemma 5.25. *For every PPT adversary $\mathcal{A}$ running Game 4, we have* $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Proof. This is a direct corollary of Lemma 4.22. □

Lemma 5.26. *We have* $\Pr[G_5(\mathcal{A})] = \Pr[G_4(\mathcal{A})]$

Proof. The two games are perfectly indistinguishable, corresponding two different samplings of randomness. □

Lemma 5.27. *If there exists a PPT adversary $\mathcal{A}$ for whom Bad_6 occurs with non-negligible probability during Game 6 as described in Theorem 5.22, there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Definition 2.42) with non-negligible probability.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce the result Bad_6 during Game 6. We show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption.

Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ simulator $\mathcal{S}$ simulates Game 6.

Setup : Simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random as well as $n \in \mathbb{N}$.
Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, \mathsf{bgp}, \mathsf{Sig}, \mathcal{R})$.

KeyGen : During the key generation simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. Then it chooses $a_i, b_i, k_i, y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for all $i \in [n]$. It sets $f_{\tau_i} = g_t^{ya_i} \cdot e(g_1, g_2^z)^{yb_i}$, $F_{\tau_i} = g_2^{a_i} \cdot (g_2^z)^{b_i}$, for all $i \in [n]$, as well as $f_{\tau_i,\tau_j} = g_t^{k_j y a_i} \cdot e(g_1, g_2^z)^{k_j y b_i}$, for all $i, j \in [n]$. It sets $h_t = e(g_1, g_2^z)$.
Additionally, it generates a key pair $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.
It sets $\mathsf{ek} = 0$ and $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_t, \{F_{\tau_i}, f_{\tau_i}\}_{i=1}^n, \{f_{\tau_i,\tau_j}\}_{i,j=1}^n)$ and gives $(\mathsf{ek}, \mathsf{vk})$ to the adversary.

Queries: Let l be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $l = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m), which collects all the label/message pairs queried by the adversary on Δ.
Moreover, whenever the l-th new dataset Δ_l is queried, $\mathcal{S}$ does the following:
If $l = \mu$, it samples a random $\zeta_\mu \in \mathbb{Z}_p$, sets $Z_\mu = (g_2^z)^{\frac{1}{\zeta_\mu}}$ and stores ζ_μ.
If $l \neq \mu$, it samples a random $\zeta_l \in \mathbb{Z}_p$ and sets $Z_l = (g_2^v)^{\frac{1}{\zeta_l}}$ and stores ζ_l. Since all Z_l are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 6. Given a query (Δ, τ, m) with $\Delta = \Delta_m$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_l} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z_l || \Delta_l)$.
If $l \neq \mu$, it samples $s_\tau, \rho_\tau \in \mathbb{Z}_p$ uniformly at random and computes
$\Lambda_\tau = \left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^m\right)^{\zeta_l}$, $R_\tau = g_1^{-(y+s_\tau)a_\tau} \cdot (g_1^r)^{\rho_\tau}$, $S_\tau = g_1^{s_\tau}$, $T_\tau = g_1^{my-k_\tau}$, $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ and gives $\sigma = (m, \sigma_{\Delta_l}, Z_l, \Lambda_\tau, R_\tau, \mathcal{T})$ to $\mathcal{A}$. We have

$$
\begin{aligned}
&e(\Lambda_\tau, Z_l) = e\left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^m, g_2^{\frac{v}{\zeta_l}}\right)^{\zeta_l} \\
&= e\left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau} \cdot (g_1^{\frac{z}{v}})^m, g_2^v\right) = g_t^{z(y+s_\tau)b_\tau + r\rho_\tau + zm} \\
&= g_t^{z(y+s_\tau)b_\tau + r\rho_\tau + zm + a_\tau(y+s_\tau) - a_\tau(y+s_\tau)} = g_t^{zm} \cdot g_t^{y(a_\tau + zb_\tau)} \cdot g_t^{-ya_\tau + r\rho_\tau} \cdot g_t^{s_\tau(a_\tau + b_\tau z)} \\
&= h_t^m \cdot f_\tau \cdot e(R_\tau, g_2) \cdot e(S_\tau, F_\tau)
\end{aligned}
$$

and this output is indistinguishable from the challenger's output during Game 6.

If $l = \mu$, simulator $\mathcal{S}$ computes $\Lambda_\tau = \left(g_1^{(y+s_\tau)b_\tau+m}\right)^{\zeta_\mu}$, $R_\tau = g_1^{-(y+s_\tau)a_\tau}$, $S_\tau = g_1^{s_\tau}$, $T_\tau = g_1^{my-k_\tau}$, $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ and gives $\sigma = (m, \sigma_{\Delta_\mu}, Z_\mu, \Lambda_\tau, R_\tau, \mathcal{T})$ to $\mathcal{A}$. We have

$$\begin{aligned} e(\Lambda_\tau, Z_\mu) &= e\left(g_1^{(y+s_\tau)b_\tau+m}, g_2^{\frac{z}{\zeta_\mu}}\right)^{\zeta_\mu} = e\left(g_1^{(y+s_\tau)b_\tau+m}, g_2^{z}\right) = g_t^{z(y+s_\tau)b_\tau+zm} \\ &= g_t^{z(y+s_\tau)b_\tau+zm+(y+s_\tau)a_\tau-(y+s_\tau)a_\tau} = g_t^{zm} \cdot g_t^{y(a_\tau+zb_\tau)} \cdot g_t^{-ya_\tau} \cdot g_t^{s_\tau(a_\tau+b_\tau z)} \\ &= h_t^m \cdot f_\tau \cdot e\left(R_\tau, g_2\right) \cdot e\left(S_\tau, F_\tau\right) \end{aligned}$$

and this output is indistinguishable from the challenger's output during Game 6.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, \sigma^*)$ be a forgery with $\sigma^* = (m^*, \sigma^*_{\Delta^*}, Z^*, \Lambda^*, R^*, \mathcal{L}^*)$ and $\mathcal{L}^* = \{(\tau, S^*_\tau)\}_{\tau\in\mathcal{I}}$, where $\mathcal{I}$ is a subset of the label space, be the forgery returned by $\mathcal{A}$.
The case of σ^* as a level-1 signature is just a simplification of the level-2 case and is omitted.
$\mathcal{S}$ computes $\hat{\sigma} \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}^*_\Delta, \boldsymbol{\sigma})$ over the signatures σ_i generated in dataset Δ^*. $\mathcal{S}$ parses $\hat{\sigma} = (\hat{m}, \sigma^*_{\Delta^*}, Z^*, \hat{\Lambda}, \hat{R}, \hat{\mathcal{L}})$ with $\hat{\mathcal{L}} = \{(\tau, \hat{S}_\tau)\}_{\tau\in\hat{\mathcal{I}}}$ where $\hat{\mathcal{I}}$ is a subset of the label space. Without loss of generality, we assume $\mathcal{I} = \hat{\mathcal{I}} = \{\tau_i\}_{i\in[n]}$, i.e. $\mathcal{I}$ is the whole label space. We can always append $\mathcal{L}$ with $(\tau_i, 1_{\mathbb{G}_1})$ and write $S_{\tau_i} = S_i$. If Game 6 outputs '1', we have $Z^* = Z_\mu$, $\Lambda^* \cdot \prod_{i=1}^n \hat{S}_i^{b_i} = \hat{\Lambda} \cdot \prod_{i=1}^n S_i^{*b_i}$, and the following hold:

$$\begin{aligned} e\left(\Lambda^*, Z_\mu\right) &= e\left(R^*, g_2\right) \cdot h_t^{m*} \cdot \prod_{i,j=1}^n f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^n f_j^{c_j} \cdot \prod_{i=1}^n e\left(S^*_{\tau_i}, F_{\tau_i}\right) \\ e\left(\hat{\Lambda}, Z_\mu\right) &= e\left(\hat{R}, g_2\right) \cdot h_t^{\hat{m}} \cdot \prod_{i,j=1}^n f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^n f_j^{c_j} \cdot \prod_{i=1}^n e\left(\hat{S}_{\tau_i}, F_{\tau_i}\right) \end{aligned}$$

Dividing those equations yields

$$\left(g_1^{(\hat{m}-m^*)z}\right)^{\frac{\zeta_\mu}{z}} = \left(\frac{R^*}{\hat{R}} \cdot \prod_{i=1}^n \frac{S_i^{*\,a_i}}{\hat{S}_i}\right)^{\frac{\zeta_\mu}{z}}$$

Thus, $\mathcal{S}$ can compute $W = g_1^{\hat{m}-m^*}$, $W' = \frac{R^*}{\hat{R}} \cdot \prod_{i=1}^n \frac{S_i^{*\,a_i}}{\hat{S}_i}$, and return W, W' as a solution to the FDHI problem. Since we have $m^* \neq \hat{m}$, we have $(W, W') \neq (1, 1)$. Our simulation has the same distribution as a real execution of Game 6.

□

Lemma 5.28. *If there exists a PPT adversary $\mathcal{A}$ who wins Game 6 with non-negligible probability, then there exists a PPT simulator $\mathcal{S}$ who can solve the FDHI problem (see Def. 2.42) with non-negligible probability.*

Proof. Assume a PPT adversary $\mathcal{A}$ wins Game 6. We show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$, $\mathcal{S}$ simulates Game 6.

Setup : Simulator $\mathcal{S}$ chooses $n \in \mathbb{N}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, \mathsf{bgp}, \mathsf{Sig}, \mathcal{R})$.

KeyGen : During the key generation simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. Then it chooses $a_i, b_i, k_i, y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for all $i \in [n]$. It sets $f_{\tau_i} = g_t^{ya_i} \cdot e(g_1, g_2^z)^{yb_i}$, $F_{\tau_i} = g_2^{a_i} \cdot (g_2^z)^{b_i}$, for all $i \in [n]$, as well as $f_{\tau_i,\tau_j} = g_t^{k_j y a_i} \cdot e(g_1, g_2^z)^{k_j y b_i}$, for all $i, j \in [n]$. It then chooses $x \in \mathbb{Z}_p$ uniformly at random. It sets $h_t = e(g_1, g_2)^x$ Additionally, it generates a key pair $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.
It sets $\mathsf{ek} = 0$ and $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_t, \{F_{\tau_i}, f_{\tau_i}\}_{i=1}^n, \{f_{\tau_i,\tau_j}\}_{i,j=1}^n)$ and gives $(\mathsf{ek}, \mathsf{vk})$ to the adversary.

Queries: Let l be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $l = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m), which collects all the label/message pairs queried by the adversary on Δ.
Moreover, whenever the l-th new dataset Δ_l is queried, $\mathcal{S}$ does the following: If $l = \mu$, it samples a random $\zeta_\mu \in \mathbb{Z}_p$, sets $Z_\mu = (g_2^z)^{\frac{1}{\zeta_\mu}}$ and stores ζ_μ. If $l \neq \mu$, it samples a random $\zeta_l \in \mathbb{Z}_p$ and sets $Z_l = (g_2^v)^{\frac{1}{\zeta_l}}$ and stores ζ_l. Since all Z_l are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 6. Given a query (Δ, τ, m) with $\Delta = \Delta_m$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_l} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z_l || \Delta_l)$.
If $l \neq \mu$, it samples $\rho_\tau, s_\tau \in \mathbb{Z}_p$ uniformly at random and computes $\Lambda_\tau = \left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau}\right)^{\zeta_l}$, $R_\tau = g_1^{-mx} \cdot g_1^{-(y+s_\tau)a_\tau} \cdot (g_1^r)^{\rho_\tau}$, $S_\tau = g_1^{s_\tau}$, $T_\tau = g_1^{mxy-k_\tau}$, $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ and gives $\sigma = (m, \sigma_{\Delta_l}, Z_l, \Lambda_\tau, R_\tau, \mathcal{T})$ to $\mathcal{A}$. We have

$$
\begin{aligned}
&e(\Lambda_\tau, Z_l) = e\left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau}, g_2^{\frac{v}{\zeta_l}}\right)^{\zeta_l} = e\left((g_1^{\frac{z}{v}})^{(y+s_\tau)b_\tau} \cdot (g_1^{\frac{r}{v}})^{\rho_\tau}, g_2^v\right) \\
&= g_t^{z(y+s_\tau)b_\tau + r\rho_\tau} = g_t^{z(y+s_\tau)b_\tau + r\rho_\tau + ym - ym + a_\tau(y+s_\tau) - a_\tau(y+s_\tau)} \\
&= g_t^{ym} \cdot g_t^{y(a_\tau + zb_\tau)} \cdot g_t^{-ym - ya_\tau + r\rho_\tau} \cdot g_t^{s_\tau(a_\tau + zb_\tau)} = h_t^m \cdot f_\tau \cdot e(R_\tau, g_2) \cdot e(S_\tau, F_\tau)
\end{aligned}
$$

and this output is indistinguishable from the challenger's output during Game 6.
If $l = \mu$, simulator $\mathcal{S}$ computes $\Lambda_\tau = \left(g_1^{(y+s_\tau)b_\tau}\right)^{\zeta_\mu}$, $R_\tau = g_1^{-mx-(y+s_\tau)a_\tau}$, $S_\tau = g_1^{s_\tau}$, $T_\tau = g_1^{mxy-k_\tau}$, $\mathcal{T} = \{(\tau, S_\tau,$

$T_\tau)\}$ and gives $\sigma = (m, \sigma_{\Delta_l}, Z_l, \Lambda_\tau, R_\tau, \mathcal{T})$ to $\mathcal{A}$. We have

$$\begin{aligned} e(\Lambda_\tau, Z_\mu) &= e\left(g_1^{(y+s_\tau)b_\tau}, g_2^{\frac{z}{\zeta_\mu}}\right)^{\zeta_\mu} = e\left(g_1^{(y+s_\tau)b_\tau}, g_2^{z}\right) = g_t^{z(y+s_\tau)b_\tau} \\ &= g_t^{z(y+s_\tau)b_\tau + xm - xm + (y+s_\tau)a_\tau - (y+s_\tau)a_\tau} = g_t^{mx} \cdot g_t^{y(a_\tau + zb_\tau)} \cdot g_t^{-mx - ya_\tau} \cdot g_t^{s_\tau(a_\tau + zb_\tau)} \\ &= h_t^m \cdot f_\tau \cdot e\left(R_\tau, g_2\right) \cdot e\left(S_\tau, F_\tau\right) \end{aligned}$$

and this output is indistinguishable from the challenger's output during Game 6.

Forgery: Let $(\mathcal{P}^*{}_{\Delta^*}, \sigma^*)$ be a forgery with $\sigma^* = (m^*, \sigma^*_{\Delta^*}, Z^*, \Lambda^*, R^*, \mathcal{L}^*)$ and $\mathcal{L}^* = \{(\tau, S^*_\tau)\}_{\tau \in \mathcal{I}}$ where $\mathcal{I}$ is a subset of the label space, be the forgery returned by $\mathcal{A}$. The case of σ^* as a level-1 signature is just a simplification of the level-2 case and is omitted.

$\mathcal{S}$ computes $\hat{\sigma} \leftarrow \mathsf{HEval}(\mathsf{pk}, \mathcal{P}^*_\Delta, \boldsymbol{\sigma})$ over the signatures σ_i generated in dataset Δ^*. $\mathcal{S}$ parses $\hat{\sigma} = (\hat{m}, \sigma^*_{\Delta^*}, Z^*, \hat{\Lambda}, \hat{R}, \hat{\mathcal{L}})$ with $\hat{\mathcal{L}} = \{(\tau, \hat{S}_\tau)\}_{\tau \in \hat{\mathcal{I}}}$ where $\hat{\mathcal{I}}$ is a subset of the label space. Without loss of generality, we assume $\mathcal{I} = \hat{\mathcal{I}} = \{\tau_i\}_{i \in [n]}$, i.e. $\mathcal{I}$ is the whole label space. We can always append $\mathcal{L}$ with $(\tau_i, 1_{\mathbb{G}_1})$, and write $S_{\tau_i} = S_i$.

If Game 6 outputs '1', we have $Z^* = Z_\mu$, $\Lambda^* = \hat{\Lambda}$, as well as

$$e\left(\Lambda^*, Z_\mu\right) = e\left(R^*, g_2\right) \cdot h_t^m \cdot \prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^{n} f_j^{c_j} \cdot \prod_{i=1}^{n} e\left(S^*_{\tau_i}, F_{\tau_i}\right) \quad \text{and}$$

$$e\left(\hat{\Lambda}, Z_\mu\right) = e\left(\hat{R}, g_2\right) \cdot h_t^m \cdot \prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^{n} f_j^{c_j} \cdot \prod_{i=1}^{n} e\left(\hat{S}_{\tau_i}, F_{\tau_i}\right).$$

Dividing those equations yields

$$\begin{aligned} \frac{\Lambda^*}{\hat{\Lambda}} &= \left(\frac{R^*}{\hat{R}} \cdot g_1^{x(m^* - \hat{m})} \cdot \prod_{i=1}^{n} \frac{S_i^{*\,a_i + b_i z}}{\hat{S}_i}\right)^{\frac{\zeta_\mu}{z}} \\ &= \left(\frac{R^*}{\hat{R}} \cdot g_1^{x(m^* - \hat{m})} \cdot \prod_{i=1}^{n} \frac{S_i^{*\,a_i}}{\hat{S}_i}\right)^{\frac{\zeta_\mu}{z}} \cdot \prod_{i=1}^{n} \frac{S_i^{*\,b_i \zeta_\mu}}{\hat{S}_i} \end{aligned}$$

Thus $\mathcal{S}$ can compute $W = \left(\frac{R^*}{\hat{R}} \cdot g_1^{x(m^* - \hat{m})} \cdot \prod_{i=1}^{n} \frac{S_i^{*\,a_i}}{\hat{S}_i}\right)^{\zeta_\mu}$, $W' = \frac{\Lambda^*}{\hat{\Lambda}} \cdot \prod_{i=1}^{n} \frac{S_i^{*\,-b_i \zeta_\mu}}{\hat{S}_i}$, and return W, W' as a solution to the FDHI problem. Since we have $\mathsf{bad}_6 = \mathsf{false}$, we have $W' \neq 1$. Our simulation has the same distribution as a real execution of Game 6.

□

6 Adding Computational Privacy to Homomorphic Authenticators

Outsourcing data and computations to the cloud has become an increasingly important aspect of IT. Such techniques provide a higher level of efficiency and flexibility and are therefore very valuable for private and commercial users. However, they also pose new risks for data security. Thus, secure outsourcing is a highly relevant research field. Cloud technologies must ensure that no malicious party gets access to the outsourced data and that no unauthorized modifications can be performed, i.e. the solutions must provide confidentiality and correctness. Both security goals can be provided by encrypting and, respectively, signing the data before outsourcing it to the cloud.

To allow for computations on the outsourced data, encryption and signature schemes with homomorphic properties were developed. However, so far most works focused on either confidentiality - provided by homomorphic encryption, or correctness - provided by homomorphic authenticators. In the previous chapter we presented such homomorphic authenticators and showed that our schemes already achieve input privacy with respect to the verifier. In this chapter we will now consider computational privacy with respect to the server, by combining homomorphic encryption with homomorphic signatures.

Catalano et al. [43] already developed a framework called “linearly homomorphic authenticated encryption with public verifiability" (LAEPuV) that allows to combine both primitives into one unified solution. They show that their framework can be instantiated with the Paillier cryptosystem [84] and any linearly homomorphic signature scheme supporting the same message space. However, the instantiation provided in [43] suffered from false negatives, i.e. correct results were considered incorrect by the verification algorithm. In this chapter, we present the first correct instantiations of LAEPuV schemes.

Contribution. In this chapter we propose novel LAEPuV instantiations, one based on the RSA problem and another based on the CDH problem. Following the methodology of Catalano et al. [43] we do so by first constructing suitable

linearly homomorphic signature schemes and afterwards applying the transformation described in [43] to them. The LAEPuV schemed derived from our CDH based homomorphic signature scheme is furthermore the first LAEPuV scheme to be context hiding, even in an information-theoretic sense.

Organization. This chapter is structured as follows. We first present a linearly homomorphic signature scheme based on the strong RSA problem in Sec. 6.1 and prove its correctness and security properties. We then describe a combination of this scheme with Paillier encryption [84], that adds computational input and output privacy with respect to the servers to the homomorphic signature scheme in Sec. 6.2. Afterwards we present a linearly homomorphic signature scheme based on the CDH problem in Sec. 6.3 and prove its correctness and security properties. Finally we show how to combine this scheme with with Paillier encryption in order to achieve computational input and output privacy with respect to the servers in Sec. 6.4.

Publications. This chapter is based on publications [S5] and [S6].

Related Work. Catalano et al. [43] proposed a framework and an instantiation for a linearly homomorphic authenticated encryption scheme providing public verifiability. As pointed out in [98], the candidate instantiation of [43] suffers from false negatives. In this work we further improve their instantiation by providing instantiations with provable correctness. We furthermore present the first construction (see Con. 6.13) that is context hiding and thus achieves information-theoretic input privacy with respect to the verifier.

6.1 An RSA Based Linearly Homomorphic Signature Scheme

Catalano et al. [43] introduced a cryptographic primitive called LAEPuV that allows a data owner to outsource encrypted data and computations on this data to the cloud. For this to be secure, the cloud must keep the data received confidential and provide measures that allow verifying the integrity of the computation results. Optimally, the results are publicly verifiable, enabling third parties such as external auditors to perform these checks.

To ensure privacy, the data owner can encrypt her data using a homomorphic encryption scheme. Due to its homomorphic properties, functions can be evaluated over the messages by evaluating corresponding functions over the ciphers. This allows the data owner to outsource the computations to a cloud such that it neither

learns the input nor the result. However, the data owner has to trust that the cloud evaluates the functions correctly.

To ensure the correctness of the result, the data owner could authenticate her data using a homomorphic authenticator scheme before outsourcing it to the cloud. This allows the data owner to delegate computations such that the data owner, or - in case of homomorphic signatures - any third party on behalf of the data owner, can verify the correctness of the computations. However, without using an encryption scheme to encrypt the data, the cloud would learn the input and the output of the computations. If both correctness and privacy are desired both schemes must be combined. More precisely, the data owner encrypts her data, signs the ciphers, and asks the cloud to evaluate the function over the ciphers. When the data owner receives the resulting cipher along with its (homomorphically computed) signature from the cloud, it can verify the computation using the signature and obtain the message by decrypting the cipher.

A naive combination of these primitives requires that the cipher space of the encryption scheme and the message space of the homomorphic signature scheme are equal. The message space of the Paillier cryptosystem is $\mathbb{Z}_N$, where $N = pq$ for two primes p, q of equal size while the corresponding cipher space is $\mathbb{Z}_{N^2}$. This leads to a performance problem as the homomorphic signature scheme has to support a significantly larger message space than the Paillier cryptosystem. Thus, Catalano et al. [43] proposed a method (see Construction 2.23) which allows combining the Paillier cryptosystem with a homomorphic signature scheme in a more efficient manner. Instead of signing the ciphers, the scheme masks the ciphers and signs the decrypted masked ciphers which have the same size as the original messages.

The first instantiation of a LAEPuV scheme described in [43], however was not correct as it suffered from false positives as shown in [98].

In the following we will describe a linearly homomorphic signature scheme $\mathsf{HSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ on which we will base our improved LAEPuV scheme. This scheme is a variation of a scheme presented by Catalano et al. [40].

Construction 6.1.

$\mathsf{Setup}(1^\lambda)$*: On input a security parameter* λ*, the algorithm chooses an integer* n *and an integer* $N_M \leftarrow p_M q_M$ *(for primes* p_M, q_M *of size* $\lambda/2$*). Additionally, it fixes a regular signature scheme* $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$*. It outputs the public parameters* $\mathsf{pp} = (\lambda, n, N_M, \mathsf{Sig})$*.*

$\mathsf{KeyGen}(\mathsf{pp})$ *: On input public parameters* pp *the algorithm chooses two (safe) primes* p_S, q_S *such that* $\gcd(N_M, \phi(N_S)) = 1$*, where* $N_S \leftarrow p_S q_S$*. It chooses* $n + 2$ *elements* $g_0, g_1, h_1, ..., h_n \xleftarrow{\$} \mathbb{Z}^*_{N_S}$ *uniformly at random and generates a key pair* $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*. It sets* $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, , p_S, q_S)$*,* $\mathsf{ek} = 0$ *and* $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, N_S, g_0, g_1, h_1, ..., h_n)$*. It outputs* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$**:** *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* $\tau \in \mathcal{T}$*, and a message* $m \in \mathbb{Z}_{N_M}$*, if* Δ *is used for the first time, it chooses a not yet used prime* e *of length* $l < \lambda/2$ *such that* $\gcd(eN_M, \phi(N_S)) = 1$*, computes its signature* $\sigma_e \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \Delta||e)$*, and stores* (Δ, e, σ_e) *in the list* L_Δ*. Otherwise, it takes* (Δ, e, σ_e) *from the list* L_Δ*. Then, it chooses* $s \xleftarrow{\$} \mathbb{Z}_{eN_M}$*, computes* x *such that* $x^{eN_M} = g_0^s h_\tau g_1^m \mod N_S$*, as well as* $e^{-1} \mod N_S$*,* $g_e = g_1^{e^{-1}}$ *and returns the signature* $\sigma = (m, e, g_e, \sigma_e, s, x)$*.*

$\mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$**:** *On input a public evaluation key* ek*,a linear function* f *given by its coefficient vector* $(f_1, \ldots, f_n)$*, and signatures* $\sigma_1, \ldots, \sigma_n$*, where* $\sigma_i = (m_i, e_i, g_{e_i}, \sigma_{e_i}, s_i, x_i)$*, the algorithm checks if the signatures share the same primes, i.e. if* $e_1 = e_i$ *for all* $i \in [n]$*. If true, the algorithm proceeds as follows, otherwise, it aborts. It sets* $e = e_1$*,* $\sigma_e = \sigma_{e_1}$*,* $g_e = g_{e_1}$ *and computes*

$$m \leftarrow \sum_{i=1}^{n} f_i m_i \mod N_M \qquad m' \leftarrow \left(\sum_{i=1}^{n} f_i m_i - m\right)/(N_M)$$

$$s \leftarrow \sum_{i=1}^{n} f_i s_i \mod eN_M \qquad s' \leftarrow \left(\sum_{i=1}^{n} f_i s_i - s\right)/(eN_M)$$

$$x \leftarrow \frac{\prod_{i=1}^{n} x_i^{f_i}}{g_0^{s'} g_e^{m'}} \mod N_S$$

and returns the signature $\sigma = (m, e, g_e, \sigma_e, s, x)$*.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$**:** *On input a public evaluation key* vk*, a multi-labeled program* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$ *, a message* m*, and a signature* σ*, the algorithm parses (without loss of generality)* $\sigma = (m, e, g_e, \sigma_e, s, x)$*. It checks whether*

$$\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_e, \Delta||e) = 1$$
$$m, s \in \mathbb{Z}_{eN_M}$$
$$x^{eN_M} = g_0^s \prod_{i=1}^{n} h_{\tau_i}^{f_i} g_1^m \mod N_S$$

If all checks pass it outputs '1', else it outputs '0'.

Note that compared to the construction of [40], our scheme has an additional component g_e and does not use the specific hash function described in [40].

In the following we show that this construction is indeed correct. For homomorphic authenticators, correctness naturally comes in two forms. On the one hand, freshly generated authenticators, obtained by using the data owner's secret key should be verified. On the other hand, authenticators derived by using the homomorphic properties should also be verified.

Proposition 6.2. *The linearly homomorphic signature scheme 6.1 achieves authentication correctness in the sense of Def. 2.5.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, $\tau \in \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $m \in \mathbb{Z}_{N_M}$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. We parse $\sigma = (m, e, g_e, \sigma_e, s, x)$. We consider the labeled identity program $\mathcal{I}_{(\tau,\Delta)}$. Here we have $f_\tau = 1$ and $f_i = 0$ for all other identifiers. By assumption it holds that Sig is a correct signature scheme and thus we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_e, \Delta||e) = 1$. By construction it holds that $s \in \mathbb{Z}_{eN_M}$ and $m < N_M$, thus $m \in \mathbb{Z}_{eN_M}$. Furthermore, it holds that

$$x^{eN_M} = g_0^s h_\tau g_1^m = g_0^s h_\tau \prod_{\tau_j \neq \tau} h_{\tau_j}^0 \cdot g_1^m = g_0^s \prod_{i=1}^{n} h_{\tau_i}^{f_i} g_1^m$$

which yields $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_{(\tau,\Delta)}, m, \sigma) = 1$.

□

Proposition 6.3. *The linearly homomorphic signature scheme 6.1 achieves evaluation correctness in the sense of Def. 2.6 if* Sig *is a correct signature scheme.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier. Let $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$ be any set of program/message/authenticator triples, such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, m_i, \sigma_i) = 1$ and $g : \mathbb{Z}_{N_M}^N \to \mathbb{Z}_{N_M}$ be an arbitrary linear function given by its coefficient vector $(c_1, \ldots, c_N)$. Let $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, \{\sigma_i\}_{i\in[N]})$.

We parse $\sigma^* = (m^*, e^*, g_e^*, \sigma_e^*, s^*, x^*)$.

By assumption we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{e_i}, \Delta||e_i) = 1$ for all $i \in [N]$. Since we have by construction $e^* = e_1$ and $\sigma_e^* = \sigma_{e_1}$ we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta^*, \Delta||e^*) = 1$.

During Eval, both m and s are reduced modulo N_M and eN_M, respectively, hence

$m, s \in \mathbb{Z}_{eN_M}$. It holds that

$$
\begin{aligned}
x^{eN_M} &= \frac{(\prod_{i=1}^{N} x_i^{c_i})^{eN_M}}{(g_0^{s'} g_e^{m'})^{eN_M}} = \frac{(\prod_{i=1}^{N} x_i^{eN_M})^{c_i}}{(g_0^{s'} g_e^{m'})^{eN_M}} = \frac{\prod_{i=1}^{N} (g_0^{s_i} h_i g_1^{m_i})^{c_i}}{(g_0^{s'} g_e^{m'})^{eN_M}} \\
&= \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{\sum_{i=1}^{N} c_i m_i}}{g_0^{s' eN_M} \cdot g_1^{\frac{m'}{e} eN_M}} = \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{\sum_{i=1}^{N} c_i m_i}}{g_0^{s' eN_M} \cdot g_1^{m' N_M}} \\
&= \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{\sum_{i=1}^{N} c_i m_i}}{\left(g_0^{(\sum_{i=1}^{N} c_i s_i - s)/(eN_M)}\right)^{eN_M} \cdot \left(g_1^{((\sum_{i=1}^{N} c_i m_i - m)/(N_M))}\right)^{N_M}} \\
&= \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{\sum_{i=1}^{N} c_i m_i}}{g_0^{\sum_{i=1}^{N} c_i s_i - s} \left(g_1^{(\sum_{i=1}^{N} c_i m_i - m)/(N_M)}\right)^{N_M}} \\
&= \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{e \sum_{i=1}^{N} c_i m_i}}{g_0^{\sum_{i=1}^{N} c_i s_i - s} g_1^{(\sum_{i=1}^{N} c_i m_i - m)}} \\
&= \frac{g_0^{\sum_{i=1}^{N} c_i s_i} \prod_{i=1}^{N} h_i^{c_i} g_1^{e \sum_{i=1}^{N} c_i m_i}}{g_0^{\sum_{i=1}^{N} c_i s_i - s} g_1^{(\sum_{i=1}^{N} c_i m_i) - m}} = g_0^{s} \prod_{i=1}^{N} h_i^{c_i} g_1^{m}
\end{aligned}
$$

which yields $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{\Delta}^{*}, m^{*}, \sigma^{*}) = 1$. □

Next, we prove that this scheme is actually unforgeable, so a malicious cloud server cannot produce an incorrect result and corresponding authenticator that will be accepted by a verifier. In order to show this we describe a security reduction between multiple security games. In the individual games we will address specific types of forgery and subsequently show that each can only be achieved with negligible probability.

The security of the scheme is given in the following theorem.

Theorem 6.4. *If* Sig *is an unforgeable signature scheme, the factorization assumption (see Def 2.43) and the strong-RSA assumption (see Def. 2.45) hold, then Construction 6.1 is secure against chosen message attacks according to Definition 2.11.*

Proof. To prove this Theorem we define a series of games with the adversary $\mathcal{A}$ and we will show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1', only with negligible probability. We write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We will make use of flag values bad_i initially set to false.

If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let $\mathsf{Bad_i}$ denote the event that bad_i is set to true during a game.

As shown in Proposition 2.16, any adversary who outputs a Type 3 forgery (see Def. 2.9) can be converted into one that outputs a Type 2 forgery. Hence we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ (see Def. 2.9) where $\mathcal{A}$ only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 2 except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 6.5, we show that any adversary $\mathcal{A}$, such that $\Pr[\mathsf{Bad}_2]$ is non negligible implies a solver for the strong RSA problem (See Def. 2.45). It is worth noticing that after this change the game never outputs '1'if $\mathcal{A}$ returns a Type 1 forgery.

Game 3 is defined as Game 3, except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_3 = \mathsf{true}$. In Lemma 6.6 we show that $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_3(\mathcal{A})]$.

Game 4 is defined as Game 4, except for the following change. When given a forgery $(\mathcal{P}^*{}_{\Delta^*}, m^*, \sigma^*)$ where $\mathcal{P}^*{}_{\Delta^*} = ((f^*, \tau_1, \ldots, \tau_n), \Delta^*)$ the simulator parses $\sigma^* = (m^*, e^*, g^*_e, \sigma^*_e, s^*, x^*)$. It computes $\hat{x} = \prod_{i=1}^n x_i^{f_i}$ and checks wheter $x^* = \hat{x}$ If it does it sets $\mathsf{bad}_4 = \mathsf{true}$.

In Lemma 6.7, we show that any adversary $\mathcal{A}$, such that $\Pr[\mathsf{Bad}_4]$ is non negligible, implies a solver for the factorization problem of safe primes.

Finally, in Lemma 6.8, we show how a simulator can use an adversary winning Game 4.

□

Lemma 6.5. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_2(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.20.

□

Lemma 6.6. *For every PPT adversary $\mathcal{A}$ running Game 3, we have* $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_3(\mathcal{A})]$.

Proof. First, $\Pr[G_3(\mathcal{A})] = \Pr[G_3(\mathcal{A}) \ \wedge \ \mathsf{bad}_3 = \mathsf{false}] = \Pr[G_3(\mathcal{A}) \mid \mathsf{bad}_3 = \mathsf{false}] \cdot \Pr[\mathsf{bad}_3 = \mathsf{false}]$, since Game 3 will always output '0' when Bad_3 occurs. Second, observe that when Bad_3 does not occur, i.e. $\mathsf{bad}_3 = \mathsf{false}$, the challenger guessed

the dataset Δ^* correctly and the outcome of Game 3 is identical to the outcome of Game 3. Since μ is chosen uniformly at random and is completely hidden to $\mathcal{A}$, we have $\Pr[\mathsf{bad}_3 = \mathsf{false}] = \frac{1}{Q}$ and therefore $\Pr[G_3(\mathcal{A})] = \frac{1}{Q}\Pr[G_3(\mathcal{A})]$. □

Lemma 6.7. *A PPT adversary $\mathcal{A}$ running Game 4, that can produce a forgery with $\mathsf{bad}_4 = \mathsf{true}$ implies a solver of the factorization problem of safe primes.*

Proof. As our Construction 6.1 is a variation of a scheme presented by Catalano et al. [40], this proof will follow the general outline of [40]. We show how to construct a simulator $\mathcal{S}$ which uses an efficient adversary $\mathcal{A}$, for whom Bad_4 occurs during Game 4 to solve the factorization problem for products of safe primes. The simluator $\mathcal{S}$ takes a input $N_S = p_S q_S = (2p'_M + 1)(2q'_M + 1)$ a product of two safe primes. It chooses $\xi \xleftarrow{\$} \mathsf{QR}_{N_S}$, where QR_{N_S} are the quadratic residues modulo N_S.

Setup : Let Q be the number of datasets for which the adversary queries signatures. The simulator chooses Q primes $e_1, ..., e_Q$. The simulator $\mathcal{S}$ defines the message space $N_M = p_M q_M$ by choosing primes p_M, q_M such that $\gcd(N_M, e_i) = 1$ for $i \in [Q]$. It chooses $n \in \mathbb{N}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, N_M, \mathsf{Sig})$.

KeyGen : The simulator $\mathcal{S}$ computes keys for the regular signature scheme $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It chooses $a, w_1, ..., w_n, b_1, ..., b_n$ uniformly at random from the set $[N_S^2]$. Following this, it sets $g_0 = \xi^{EN_M}$, $g_1 = \xi^{EN_M a}$, and $h_i = \xi^{EN_M(e_\mu w_i + b_i)}$ for $i \in [n]$, where $E = \prod_{i=1}^{n} e_i$. We write $E_l = \prod_{\substack{i=1 \\ i \neq l}}^{n} e_i$.
Then, it gives $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, N_S, g_0, g_1, h_1, ..., h_n)$ to the adversary.
Since $a, b_1, ..., b_n$ are chosen uniformly at random, the adversary can not distinguish vk from a public key of the genuine scheme.

Queries: For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m), which collects all the label/message pairs queried by the adversary on Δ. On query (Δ_l, τ, m) the simulator $\mathcal{S}$ checks if $l = \mu$. We distinguish between two cases.

- **Case 1**: The adversary queries the signatures for the l-th dataset, where $l \in [n] \backslash \{\mu\}$.
- **Case 2**: The adversary queries the signatures for the μ-th dataset.

Case 1: On query (Δ_l, τ, m), $\mathcal{S}$ computes $\sigma_{e_l} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_l || e_l)$. It computes $g_{e_l} = \xi^{E_l N_M a_1}$, chooses $s \xleftarrow{\$} \mathbb{Z}_{e_\mu N_M}$, and sets
$x = \left(\xi^{E_l}\right)^{s + e_\mu w_\tau + b_\tau + am}$. It returns $\sigma = (m, e_l, g_{e_l}, \sigma_{e_l}, s, x)$ to $\mathcal{A}$. To verify

that this is a valid signature, note that

$$\begin{aligned} x^{e_l N_M} &= \left(\xi^{E_l}\right)^{(s+e_\mu w_\tau + b_\tau + am)e_l N_M} \\ &= \left(\xi^{N_M E}\right)^{s+e_\mu w_\tau + b_\tau + am} \\ &= \xi^{N_M E s} \cdot \xi^{N_M E(e_\mu w_\tau + b_\tau)} \cdot \xi^{N_M E(am)} \\ &= g_0^s \cdot h_\tau \cdot g_1^m \end{aligned}$$

Case 2: For a query in the dataset Δ_μ for which the simulator $\mathcal{S}$ guessed that $\mathcal{A}$ will produce a forgery, $\mathcal{S}$, computes $\sigma_{e_\mu} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \Delta_\mu || e_\mu)$. It computes $g_{e_\mu} = \xi^{E_\mu N_M a_1}$. It sets $s = (b_\tau - am) \mod e_\mu$ and sets $x = \xi^{E \cdot y}$, where $y = \frac{b_\tau - s + e_\mu w_\tau - am}{e_\mu}$. It holds that

$$\begin{aligned} x^{e_\mu N_M} &= \xi^{E \cdot y \cdot e_\mu \cdot N_M} \\ &= \xi^{N_M E(\frac{b_\tau - s + e_\mu w_\tau - am}{e_\mu}) \cdot e_\mu} \\ &= \xi^{N_M E(b_\tau - s + e_\mu w_\tau - am)} \\ &= \xi^{N_M E s} \cdot \xi^{N_M E_\mu (e_\mu w_\tau + b_\tau)} \cdot \xi^{N_M E(am)} \\ &= g_0^s \cdot h_\tau \cdot g_1^m \end{aligned}$$

thus, $\sigma = (m, e, g_e, \sigma_e, s, x)$ is a valid signature on m.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by the adversary $\mathcal{A}$. Parse $\sigma^* = (m^*, e, \sigma_e, s, x)$. By assumption we have $x^* = \hat{x}$, where $\hat{x} = \prod_{i=1}^n x_i^{f_i}$. We compute $m' = \sum_{i=1}^n f_i m_i$ as well as $s' = \sum_{i=1}^n f_i s_i$. Since σ^* is forgery we have $(x^*)^{e_\mu N_M} = g_0^{s^*} \prod_{i=1}^n h_i^{f_i} g_1^{m^*} = \left(\xi^{N_M E_\mu}\right)^{s^* + \sum_{i=1}^n (e_\mu w_{\tau_i} + b_{\tau_i}) f_i + am^*}$. Furthermore, it holds that

$$\begin{aligned} 1 = \left(\frac{x^*}{\prod_{i=1}^n x_i^{f_i}}\right)^{e_\mu N_M} &= \frac{g_0^{s^*} \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^{m^*}}{\prod_{i=1}^n (g_0^{s_i} \cdot h_{\tau_i} \cdot g_1^{m_i})^{f_i}} \\ &= \frac{\left(\xi^{N_M E(s^* + (e_\mu w_\tau) f_i + am^*)}\right)}{\left(\xi^{N_M E(\sum_{i=1}^{nk} f_i s_i + (e_\mu w_\tau) f_i + a \sum_{i=1}^n f_i m_i)}\right)} \\ &= \xi^{N_M E(s^* - s' + a(m^* - m'))} \end{aligned}$$

Let $y = N_M E(s^* - s' + a(m' - m'))$. We have found an integer y such that $y = 0 \mod \varphi(N_S)$. Notice that we can write $a = \alpha p'_S q'_S + \beta$ with $0 \leq \beta \leq p'_S q'_S$ and α is information-theoretically hidden from $\mathcal{A}$. Therefore we have $y \neq 0$ with non-negligible probability and can use the knowledge of $\varphi(N_S)$ to reconstruct $p_S q_S = (2p'_S + 1)(2q'_S + 1)$.

□

Lemma 6.8. *A PPT adversary $\mathcal{A}$ that wins Game 4 implies a solver of the strong RSA problem (see Def. 2.45).*

Proof. This proof works analogously to Case II-b presented by Catalano et al. [39, Theorem 2]. The main difference is the presence of the component g_e in our authenticators σ. We will describe how to simulate such a component. The simulator $\mathcal{S}$ takes as input a strong RSA instance $\xi, N_S = p_S q_S$ (for safe primes p_S, q_S) (see Def. 2.45). During key generation $\mathcal{S}$ samples $y \xleftarrow{\$} \mathsf{QR}_{N_S}$ and sets $g_1 = y^E$, where we have again $E = \prod_{i=1}^{n} e_i$. For a query in dataset Δ_l, $\mathcal{S}$ can compute $g_e = y^{E_l}$, where we again have $E_l = \prod_{\substack{i=1 \\ i \neq l}}^{n} e_i$. The rest of the proof is a corollary of [39, Theorem 2].

□

6.2 RSA Based Linearly Homomorphic Authenticated Encryption

We are now ready to provide our new scheme, which simultaneously achieves verifiability and computational input and output privacy with respect to the servers. This is achieved by applying the transformation of [43] to our Construction 6.1.

Construction 6.9.

RSA-based LAEPuV:

$\mathsf{AKeyGen}(1^\lambda, n)$***:*** *On input a security parameter λ and an integer n, the algorithm samples the four (safe) primes p_E, q_E, p_S, q_S, the group elements $g_0, g_1, h_1, ..., h_n \in \mathbb{Z}^*_{N_S}$ and $g \in \mathbb{Z}^*_{N_E^2}$ of order N_E, and the hash function $H \in \mathcal{H} : \{0,1\}^* \to \mathbb{Z}^*_{N_E^2}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen_{Sig}}, \mathsf{Sign_{Sig}}, \mathsf{Ver_{Sig}})$. It computes keys for the regular signature scheme $(\mathsf{sk_{Sig}}, \mathsf{pk_{Sig}}) \leftarrow \mathsf{KeyGen_{Sig}}(1^\lambda)$ and returns the key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = (p_E, q_E, p_S, q_S, g, \mathsf{sk_{Sig}})$ and $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_n, H, \mathsf{pk_{Sig}})$. It returns the key pair $(\mathsf{sk}, \mathsf{pk})$.*

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$***:*** *On input a secret key sk, a dataset identifier Δ, an input identifier $\tau \in \mathcal{T}$, and a message $m \in \mathbb{Z}^T_{N_E}$, it chooses $\beta \xleftarrow{\$} \mathbb{Z}_{N_E^2}$ uniformly at random . It computes the ciphertext $C \leftarrow g^m \cdot \beta^N \mod N_E^2$, computes $S \leftarrow H(\Delta||\tau)$ and computes $(a, b) \in \mathbb{Z}_{N_E} \times \mathbb{Z}^*_{N_E}$ such that $g^a \cdot b^{N_E} = C \cdot S \mod N_E^2$ using the factorization of N_E (see [84] for a detailed description). If Δ is used the first time, it chooses a not yet used prime e of length $l < \lambda/2$ such that $\gcd(eN_E, \phi(N_S)) = 1$, computes $g_e = g_1^{e^{-1}}$ and its signature*

$\sigma_e \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \tau||e)$, *and stores* (Δ, e, σ_e) *in the list* L_Δ. *Otherwise, it takes* (Δ, e, σ_e) *from the list* L_Δ. *Then, it chooses* $s \xleftarrow{\$} \mathbb{Z}_{eN_E}$, *computes the value* x *such that* $x^{eN_E} = g_0^s h_\tau g_1^a \mod N_S$, *and returns the ciphertext* $c = (C, a, b, e, g_e, \sigma_e, s, x)$.

$\mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^n)$**:** *On input a public key* pk, *a multi-labeled program* $\mathcal{P}_\Delta$, *and a set of ciphertexts* c_i, *it parses* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$, *where* f *is a linear function given by its coefficient vector* $(f_1, \ldots, f_n)$, *as well as* $c_i = (C_i, a_i, b_i, e_i, g_{e_i}, \sigma_{e_i}, s_i, x_i)$ *It checks whether* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{e_i}, \Delta||e_i) = 0$ *for any* $i \in [n]$. *Furthermore, the algorithm checks if there are two indexes* $i \neq j \in [n]$ *such that* $e_i \neq e_j$. *If one of the checks is true, the algorithm aborts. Otherwise, the algorithm sets* $e = e_1$, $g_e = g_{e_1}$, $\sigma_e = \sigma_{e_1}$, *computes*

$$C = \prod_{i=1}^n C_i^{f_i} \mod N_E^2 \qquad a = \sum_{i=1}^n f_i a_i \mod N_E$$

$$b = \prod_{i=1}^n b_i^{f_i} \mod N_E^2 \qquad s = \sum_{i=1}^n f_i s_i \mod eN_E$$

$$s' = \left(\sum_{i=1}^n f_i s_i - s\right)/(eN_E) \qquad x = \frac{\prod_{i=1}^n x_i^{f_i}}{g_0^{s'}} \mod N_S$$

$$a' = \left(\sum_{i=1}^n f_i a_i - a\right)/N_E \qquad x = \frac{\prod_{i=1}^n x_i^{f_i}}{g_0^{s'} g_e^{a'}} \mod N_S$$

Then, it returns the ciphertext $c = (C, a, b, e, g_e, \sigma_e, s, x)$.

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$**:** *On input a public key* pk, *a multi-labeled program* $\mathcal{P}_\Delta$, *and a ciphertext* c, *it parses* $\mathcal{P}_\Delta = ((f_1, ..., f_n), \tau_1, \ldots, \tau_n, \Delta)$ *and* $c = (C, a, b, e, g_e, \sigma_e, s, x)$. *The algorithm checks whether the following equations hold:*

$$\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_e, \Delta||e) = 1$$

$$a, s \in \mathbb{Z}_{eN_E}$$

$$x^{eN_E} = g_0^s \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^a \mod N_S$$

$$g^a b^{N_E} = C \prod_{i=1}^n H(\Delta||\tau_i)^{f_i} \mod N_E^2$$

If all four checks pass, the algorithm returns '1', i.e. c *is a valid ciphertext. Otherwise, it returns '0', i.e.* c *is an invalid ciphertext.*

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$**:** *On input a secret key* sk, *a multi-labeled program* $\mathcal{P}_\Delta$ *and a ciphertext* c, *the algorithm does the following. It runs* $b \leftarrow \mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$. *If* $b = 0$ *it outputs* $\perp$. *Otherwise, it computes* (m, β) *such that* $g^m \beta^{N_E} = C \mod N_E^2$ *and returns* m.

We will explicitly show this combination of primitives is still correct. The first instantiation of a LAEPuV scheme described in [43] was indeed not correct as it suffered from false negatives as shown in [98]. A correct execution of Eval and a correct computation of a function could (with non-negligible probability) lead to $\mathsf{AVerify}$ not accepting the result.

Theorem 6.10. *Construction 6.9 is a correct LAEPuV scheme in the sense of Definition 2.20.*

Proof. In the following, we show that each condition described in Definition 2.20 holds. Throughout this proof, let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n)$ be a key pair, where $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, p_E, q_E, p_S, q_S)$ and $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_n, H, \mathsf{pk}_{\mathsf{Sig}})$.

Condition 1: Let $m \in \mathbb{Z}_{N_E}$ be an arbitrary message, Δ be an arbitrary dataset identifier, $\tau \in \mathcal{T}$, $c = (C, a, b, e, g_e, \sigma_e, s, x) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$ be the encryption of m, and f the linear function given by the i-th unit vector.

By construction we have $a, s \in \mathbb{Z}_{eN_E}$ and $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_e, \Delta||e) = 1$. It holds that

$$x^{eN_E} = g_0^s h_i g_1^a = g_0^s h_{\tau_i} \prod_{\substack{j=1 \\ j \neq i}}^{n} h_{\tau_j}^0 g_1^a = g_0^s \prod_{j=1}^{n} h_j^{f_j} g_1^a$$

and

$$g^a b^{N_E} = CR = CH(\Delta||\tau_i) = CH(\Delta||\tau_i) \prod_{\substack{j=1 \\ j \neq i}}^{n} H(\Delta||\tau_j)^0 = C \prod_{j=1}^{n} H(\Delta||\tau_j)^{f_j}$$

which yields $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$, for $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$. Thus, $\mathsf{ADecrypt}$ returns the Paillier decryption of C, i.e. $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = m$.

Condition 2: We prove the equivalence by showing that both implications are satisfied.

$\Leftarrow$: Let $m \in \mathcal{M} = \mathbb{Z}_{N_E}$ be a message, f be a linear function given by its coefficient vector $(f_1, ..., f_n)$ with $f_i < N_E$ for $i \in [n]$, and c be a ciphertext such that $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = m$. The fact that $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) \neq \perp$ directly leads to $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$.

$\Rightarrow$: Let $c = (C, a, b, eg_e, \sigma_e, s, x) \in \mathcal{C}$ be a ciphertext, Δ be a dataset identifier, and f be a linear function such that $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$. Since $\mathsf{ord}(g) = N_E$, this guarantees that the Paillier decryption of C yields $m \in \mathcal{M} = \mathbb{Z}_{N_E}$. Thus, $\exists m \in \mathcal{M} : \mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c) = m$.

Condition 3: Let Δ be an arbitrary dataset identifier, $m_1, ..., m_n \in \mathbb{Z}_{N_E}$ be messages, and $c_i = (C_i, a_i, b_i, e, g_{e_i}, \sigma_e, s_i, x_i) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau_i, m_i)$ be the ciphertext obtained by encrypting the message m_i for $i \in [n]$.

Let f be a linear function given by its coefficient vector $(f_1, ..., f_n)$ such that $f_i < N_E$ for all $i \in [n]$ and $c = (C, a, b, e, g_e, \sigma_e, s, x) \leftarrow \mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^n)$ be the ciphertext obtained by evaluating the function f over the ciphertexts c_i.

By construction it holds that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_e, \Delta||e) = 1$. During AEval, s and a are reduced modulo eN_E and N_E, respectively. Thus, $s, a \in \mathbb{Z}_{eN_E}$. In order to show that $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$, it remains to show that

$$x^{eN_E} = g_0^s \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^a \mod N_S$$
$$g^a b^{N_E} = C \prod_{i=1}^n H(\Delta||\tau_i)^{f_i} \mod N_E^2$$

For the first equation we have

$$\begin{aligned} x^{eN_E} &= \frac{(\prod_{i=1}^n x_i^{f_i})^{eN_E}}{(g_0^{s'} g_e^{a'})^{eN_E}} = \frac{\prod_{i=1}^n (g_0^{s_i} h_{\tau_i} g_1^{a_i})^{f_i}}{(g_0^{s'} g_1^{a'e^{-1}})^{eN_E}} \\ &= \frac{g_0^{\sum_{i=1}^n f_i s_i} \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^{\sum_{i=1}^n f_i a_i}}{\left(g_0^{(\sum_{i=1}^n f_i s_i - s)/(eN_E)} g_1^{((\sum_{i=1}^n f_i a_i - a)/(N_E))e^{-1}}\right)^{eN_E}} \\ &= \frac{g_0^{\sum_{i=1}^{nk} f_i s_i} \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^{\sum_{i=1}^n f_i a_i}}{g_0^{\sum_{i=1}^n f_i s_i - s} \left(g_1^{(\sum_{i=1}^n f_i a_i - a)/(eN_E)}\right)^{eN_E}} \\ &= \frac{g_0^{\sum_{i=1}^n f_i s_i} \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^{\sum_{i=1}^n f_i a_i}}{g_0^{\sum_{i=1}^n f_i s_i - s} g_1^{\sum_{i=1}^n f_i a_i - a}} = g_0^s \prod_{i=1}^n h_{\tau_i}^{f_i} g_1^a \end{aligned}$$

For the latter equation we obtain

$$\begin{aligned} C \prod_{i=1}^n H(\Delta||\tau_i)^{f_i} &= \prod_{i=1}^n C_i^{f_i} \prod_{i=1}^n H(\Delta||\tau_i)^{f_i} = \prod_{i=1}^n (C_i H(\Delta||\tau_i))^{f_i} \\ &= \prod_{i=1}^n (g^{a_i} b_i^{N_E})^{f_i} = g^{\sum_{i=1}^n f_i a_i} \prod_{i=1}^n b_i^{f_i N_E} = g^a b^{N_E} \end{aligned}$$

Thus, it holds that $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$.
Finally, we have $C = \prod_{i=1}^n C_i^{f_i} = \prod_{i=1}^n (g^{m_i} \beta_i^{N_E})^{f_i} = g^{\sum_{i=1}^n f_i m_i} \prod_{i=1}^n \beta_i^{f_i N_E}$. hence Paillier decryption yields $\sum_{i=1}^n f_i m_i = f(m_1, ..., m_n)$, which leads to $\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, \mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^n)) = f(m_1, ..., m_n)$. Thus, Construction 6.9 satisfies Condition 1 - 3 which proves the statement.

□

The security of this construction can be shown by applying the generic result of [43].

Theorem 6.11. *([43]) In the random oracle model, if the DCR Assumption (see Def. 2.44), the factorization assumption (see Def. 2.43), the strong RSA Assumption (see Def. 2.45) hold and H is a random oracle then Construction 6.9 is a LH-IND-CCA secure (see Definition 2.21) LAEPuV scheme.*

Proof. This is a direct corollary of Theorem 2.24 and Theorem 6.4. □

Theorem 6.12. *([43]) Construction 6.9 is a LH-Uf-CCA secure LAEPuV scheme according to Definition 2.22.*

Proof. This is a direct corollary of Theorem 2.25 and Theorem 6.4. □

6.3 A CDH Based Linearly Homomorphic Signature Scheme

In Section 6.1 we described a homomorphic authenticator scheme that can be transformed into a LAEPuV scheme and thus achieves computational input and output privacy with respect to the servers. Compared to the schemes discussed in Chapter 5 however this construction does not achieve input privacy with respect to the verifier. In the following we provide an alternative instantiation that does additionally achieve this even in an information-theoretic sense. Furthermore, while Construction 6.1 only supported linear evaluations of filed elements, i.e. vectors of length 1, we will discuss how this can be generalized to vectors of polynomial length T.

In the following we will describe a linearly homomorphic signature scheme $\mathsf{HSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ based on CDH in bilinear groups. Multi-labeled programs contain linear functions f given by their coefficients, i.e. $f = (f_1, \ldots, f_n)$.

Construction 6.13.

$\mathsf{CDH} - \mathsf{LinAuth}$*:*

$\mathsf{Setup}(1^\lambda)$*: On input a security parameter λ the algorithm runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. It chooses $n, T \in \mathbb{N}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It samples $H_1, \ldots, H_T \xleftarrow{\$} \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, T, \mathsf{bgp}, \mathsf{Sig}, \Phi, H_1, \ldots, H_T)$.*

$\mathsf{KeyGen}(\mathsf{pp})$ *: On input public parameters pp it chooses a random seed $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function Φ. It computes keys for the regular signature scheme $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It samples $R_{\tau_1}, \ldots R_{\tau_n} \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It sets $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, K)$, $\mathsf{ek} = 0$ and $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, R_{\tau_1}, \ldots R_{\tau_n})$. It outputs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$**:** *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* $\tau \in \mathcal{T}$*, and a message* $m \in \mathbb{Z}_p^T$*, the algorithm generates the parameters for the dataset identified by* Δ*, by running* $z \leftarrow \Phi_K(\Delta)$ *and computing* $Z = g_2^z$*. It binds* Z *to the dataset identifier* Δ *by using the regular signature scheme, i.e. it sets* $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$*. Then, it computes* $\Lambda \leftarrow \left(R_\tau \cdot \prod_{j=1}^T H_j^{m[j]}\right)^z$*. It outputs the signature* $\sigma = (\sigma_\Delta, Z, \Lambda)$*.*

$\mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$**:** *On input a public evaluation key* ek*, a linear function* f*, and signatures* $\sigma_1, \ldots, \sigma_n$*, where* $\sigma_i = (\sigma_{\Delta,i}, Z_i, \Lambda_i)$*, the algorithm checks if the signatures share the same public values, i.e. if* $\sigma_{\Delta,1} = \sigma_{\Delta,i}$ *and* $Z_1 = Z_i$ *for all* $i = 2, \ldots, n$ *and the signature for each set of public values is correct and matches the dataset identifier* Δ*, i.e.* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta_i||Z_i) = 1$ *for any* $i \in [n]$*. If this is not the case it outputs* $\perp$*. Else it computes* $\Lambda = \prod_{i=1}^n \Lambda_i^{f_i}$*, and returns the signature* $\sigma = (\sigma_{\Delta,1}, Z_1, \Lambda)$*.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$**:** *On input a public evaluation key* vk*, a multi-labeled program* $\mathcal{P}_\Delta$ *containing a linear function* f*, a message* m*, and a signature* σ*, the algorithm parses (without loss of generality)* $\sigma = (\sigma_\Delta, Z, \Lambda)$*. It checks whether* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$*. If not it outputs '0'. It computes* $R \leftarrow \prod_{i=1}^n R_{\tau_i}^{f_i}$ *and checks if* $e\left(R \cdot \prod_{j=1}^T h_j^{m[j]}, Z\right) = e\left(\Lambda, g_2\right)$*. If it does it outputs '1', else it outputs '0'.*

We again show correctness in its two flavors. On the one hand freshly generated authenticators, obtained by using the data owners secret key should be verified. On the other hand authenticators derived by using the homomorphic properties should also be verified.

Proposition 6.14. *Construction 6.13 achieves authentication correctness in the sense of Def- 2.5 if* Sig *is a correct signature scheme.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, $\tau \in \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $m \in \mathbb{F}_p$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. We parse $\sigma = (\sigma_\Delta, Z, \Lambda)$. We consider the labeled identity program $\mathcal{I}_{(\tau,\Delta)}$. Here we have $f_\tau = 1$ and $f_i = 0$ for all other identifiers. By construction it holds that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We then set $R = \prod_{i=1}^n R_{\tau_i}^{f_i} = R_\tau$. This yields

$$
\begin{aligned}
e\left(R_\tau \cdot \prod_{j=1}^T H_j^{m[j]}, Z\right) &= e\left(R_\tau \cdot \prod_{j=1}^T H_j^{m[j]}, g_2^z\right) \\
&= e\left(\left(R_\tau \cdot \prod_{j=1}^T H_j^{m[j]}\right)^z, g_2\right) = e\left(\Lambda, g_2\right)
\end{aligned}
$$

Thus, we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{I}_{(\tau,\Delta)}, m, \sigma) = 1$.

□

Proposition 6.15. *Construction 6.13 achieves evaluation correctness in the sense of Def- 2.6.*

Proof. Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow Setup1^\lambda$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier. Let $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$ be any set of program/message/authenticator triples, such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ and $g : \left(\mathbb{Z}_p^T\right)^N \to \mathbb{Z}_p^T$ be an arbitrary linear function given by its coefficient vector $(g_1, \ldots, g_N)$. Let $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, \{\sigma_i\}_{i\in[N]})$. We parse $\sigma^* = (\sigma^*_\Delta, Z^*, \Lambda^*)$. By construction we have $Z^* = Z_i$ for all $i \in [N]$ and $\sigma^*_\Delta = \sigma_{\Delta,1}$, hence we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma^*_\Delta, \Delta||Z^*) = 1$. To prove evaluation correctness it remains to show that $e\left(R^* \cdot \prod_{j=1}^T H_j^{m^*[j]}, Z^*\right) = e\left(\Lambda^*, g_2\right)$, where $R^* = \prod_{i=1}^N R_i^{g_i}$. It holds that

$$
\begin{aligned}
e\left(R^* \cdot \prod_{j=1}^{T} H_j^{m^*[j]}, Z^*\right) &= e\left(\prod_{i=1}^{N} R_i^{g_i} \cdot \prod_{j=1}^{T} H_j^{\sum_{i=1}^N g_i m_i[j]}, Z^*\right) \\
&= e\left(\prod_{i=1}^{N} R_i \cdot \prod_{i=1}^{N}\prod_{j=1}^{T} \left(H_j^{m_i[j]}\right)^{g_i}, Z^*\right) \\
&= \prod_{i=1}^{N} e\left(R_i \cdot \prod_{j=1}^{T} H_j^{m_i[j]}, Z_i\right)^{g_i} \\
&= \prod_{i=1}^{N} e\left(\Lambda_i, g_2\right)^{g_i} = e\left(\prod_{i=1}^{N} \Lambda_i^{g_i}, g_2\right) = e\left(\Lambda^*, g_2\right)
\end{aligned}
$$

hence we have $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, m^*, \sigma^*) = 1$.

□

Next, we discuss the efficiency properties of this construction. Succinctness guarantees that authenticators are suitably small leading to low bandwidth requirements. Efficient verification ensures that the verification time is small, in our case it is even in constant time.

Proposition 6.16. *Construction 6.13 is succinct linearly homomorphic signature scheme in the sense of Def. 2.7.*

Proof. An authenticator produced by either Auth or Eval consists of a conventional signature, one element in $\mathbb{G}_1$ and one element in $\mathbb{G}_2$. This is a constant and independent of n. Therefore our construction is succinct. □

Proposition 6.17. *Construction 6.13 allows for efficient verification in the sense of Def. 2.8.*

Proof. We describe the two algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$.

$\mathsf{VerPrep}(\mathsf{pk}, \mathcal{P})$: This algorithm parses $\mathcal{P} = ((f_1, \ldots, f_n), \tau_1, \ldots \tau_n)$ and takes the R_i for $i \in [n]$ contained in the public key. It computes $R_{\mathcal{P}} \leftarrow \prod_{i=1}^{n} R_i^{f_i}$ and outputs $\mathsf{vk}_{\mathcal{P}} = (\mathsf{pk}_{\mathsf{Sig}}, R_{\mathcal{P}})$, where $\mathsf{pk}_{\mathsf{Sig}}$ is taken from vk

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, m, \sigma, \Delta)$**:** This algorithm is analogous to Ver, except that the value $\prod_{i=1}^{n} R_{\tau_i}^{f_i}$ has been precomputed as $R_{\mathcal{P}}$.

This satisfies correctness. During EffVer, the verifier now checks whether $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$ and computes

$$e\left(R_{\mathcal{P}} \cdot \prod_{j=1}^{T} H_j^{m[j]}, Z\right) = e\left(\Lambda, g_2\right)$$

The running time of EffVer is thus independent of n. Thus our construction is constant time (in an amortized sense).

□

Theorem 6.18. *Construction 6.13 is a perfectly context hiding linearly homomorphic signature scheme in the sense of Def. 2.18 if* Sig *is a deterministic signature scheme.*

Proof. We show that our scheme is perfectly externally context hiding in the sense of Def. 2.18, by comparing the distributions of homomorphically derived signatures to that of simulated signatures. First we note that in our case the algorithm Hide is just the identity function, i.e. $\sigma \leftarrow \mathsf{Hide}(\mathsf{vk}, m, \sigma)$ for all vk, m, σ and we have $\mathsf{HideVer} = \mathsf{Ver}$. We will show how to construct a simulator Sim that outputs signatures perfectly indistinguishable from the ones obtained by running Auth and Eval. Parse the simulator's input as $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, K)$, $\mathcal{P}_\Delta = ((f_1, \ldots, f_n), \tau_1, \ldots, \tau_n, \Delta)$, and $\tilde{m} = (\tilde{m}[1], \ldots, \tilde{m}[T])$. With this information the simulator computes the following:

$Z' = g_2^z$ where $z \leftarrow \Phi_K(\Delta)$
$\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$
$\Lambda' = \left(\prod_{i=1}^{n} R_i^{f_i} \cdot \prod_{j=1}^{T} h_j^{m[j]}\right)^z$

The simulator outputs the signature $\sigma' = (\sigma'_\Delta, Z', \Lambda')$.

We will now show that this simulator allows for perfectly context hiding security. We will fix an arbitrary key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$, a multi-labeled program $\mathcal{P}_\Delta$, and messages $m_1, \ldots, m_n \in \mathbb{Z}_p^T$.

Let $\sigma \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$ and parse it as $\sigma = (\sigma_\Delta, Z, \Lambda)$.

We look at each component of the signature.

We have $Z = \Phi_K(\Delta)$ by definition and therefore also $z = z'$. In particularly we also have $Z = Z'$ where $Z = g_2^z$ and $Z' = g_2^{z'}$.

We have $\sigma_\Delta = \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$ by definition and since $Z = Z'$ therefore also $\sigma_\Delta = \sigma'_\Delta$ since Sig is deterministic.

We have $\Lambda = \prod_{i=1}^{n}(R_i \cdot \prod_{j=1}^{T} H_j^{m_i[j]})^{z \cdot f_i} = (\prod_{i=1}^{n} R_i^{f_i})^z \cdot (\prod_{j=1}^{T}\prod_{i=1}^{k} H_j^{f_i \cdot m_i[j]})^z = (\prod_{i=1}^{n} R_i^{f_i} \cdot \prod_{j=1}^{T} H_j^{m[j]})^z$, where the last equation holds since $m = \sum_{i=1}^{n} f_i \cdot m_i$. Thus we also have $\Lambda = \Lambda'$.

We can see that we have *identical* elements and therefore even a computationally unbounded distinguisher has no advantage distinguishing the two cases. □

In order to show unforgeability we again provide a security reduction over various security games. In particular we deal with special types of forgeries in one game, showing that such a specific forgery implies an algorithm for solving a certain cryptographic problem. Then, knowing that such a specific forgery can no longer be produced by the adversary we use this knowledge in order to provide the following security reduction in the following game to a different cryptographic assumption.

Theorem 6.19. *If* Sig *is an unforgeable signature scheme,* Φ *is a pseudorandom function, the DL assumption (see Def 2.36) holds in* $\mathbb{G}_1$ *and the* $co - CDH^*$ *assumption (see Def. 2.39) holds in* bgp*, then the signature scheme describe above is a weakly-unforgeable homomorphic signature scheme in the sense of Def 2.13.*

Proof. To prove this Theorem we define a series of games with the adversary $\mathcal{A}$ and we will show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1', only with negligible probability. Following the notation of [37] we will write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We will make use of flag values bad_i initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during a game.

Game 1 is the experiment $\mathsf{Weak} - \mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{HomSign}}$ (see Def. 2.12) where $\mathcal{A}$ only outputs Type-1 or Type-2 forgeries.

Game 2 is defined as Game 1 except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 6.20, , we show that Bad_2 cannot occur if Sig is unforgeable. It is worth noticing that after this change the game never outputs '1'if $\mathcal{A}$ returns a Type 1 forgery.

Game 3 is defined as Game 2, except that the keyed pseudorandom function Φ_K is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. In Lemma 6.21, we show that these two games are indistinguishable if Φ is pseudorandom.

Game 4 is defined as Game 3, except except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_4 = \mathsf{true}$. In Lemma 6.22 we show that $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Game 5 is defined as Game 4, except for the following change. When given a forgery $(\mathcal{P}^*{}_{\Delta^*}, m^*, \sigma^*)$ where $\mathcal{P}^*{}_{\Delta^*} = ((f^*, \tau_1, \ldots, \tau_n), \Delta^*)$ the simulator computes $\hat{m} = f^*(m_{1,\Delta}, \ldots, m_{n,\Delta})$. It checks whether $\prod_{j=1}^{T} H_j^{\hat{m}[j]} = \prod_{j=1}^{T} H_j^{m^*[j]}$ holds. If it does it sets $\mathsf{bad}_5 = \mathsf{true}$.
In Lemma 6.23, we show that any adversary $\mathcal{A}$, such that $\Pr[\mathsf{Bad}_5]$ is non negligible, implies a solver for the DL problem.
Finally, in Lemma 6.24, we show how a simulator can use an adversary winning Game 5 to solve the $co - CDH^*$ problem in bgp.

□

Lemma 6.20. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_2(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.20.

□

Lemma 6.21. *For every PPT adversary $\mathcal{A}$ running Game 3, there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{\Phi,\mathsf{D}}(\lambda)$.

Proof. This is a direct corollary of Lemma 4.21.

□

Lemma 6.22. *For every PPT adversary $\mathcal{A}$ running Game 4, we have* $\Pr[G_3(\mathcal{A})] = Q \cdot \Pr[G_4(\mathcal{A})]$.

Proof. This is a direct corollary of Lemma 6.22. □

Lemma 6.23. *For every PPT adversary $\mathcal{A}$ running Game 5, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_5] \leq \mathsf{Adv}^{DL}_{\mathcal{S}}(\lambda)$.

Proof. This is a direct corollary of Theorem 4.15. □

Lemma 6.24. *An efficient adversary $\mathcal{A}$ winning Game 5 in Theorem 6.19, can be used to break the CDH assumption.*

Proof. We will now show how to construct a simulator $\mathcal{S}$ which uses an efficient adversary $\mathcal{A}$ against Game 5 to solve the CDH problem. Let $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(1^\lambda)$ be a bilinear group of order p. The simulator $\mathcal{S}$ is given g_1, g_1^a, g_2^b, where $a, b \xleftarrow{\$} \mathbb{Z}_p$, and intends to compute g_1^{ab}.

Initialization: Let Q be the number of datasets in which the adversary makes signature queries. The adversary gives the simulator all messages $\{m_{(i,l)}\}_{i=1}^{n}$, for $l \in [Q]$ on which it makes signature queries.

Setup : The simulator $\mathcal{S}$ chooses $n, T \in \mathbb{N}$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It chooses $s_j \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for $j \in [T]$ and sets $H_j = (g_1^a)^{s_j}$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, T, \mathsf{bgp}, \mathsf{Sig}, \Phi, H_1, \ldots, H_T)$. Note that since the $s : j$ are uniformly sampled from $\mathbb{Z}_p$ and the H_j are uniformly random from the grpup $\mathbb{G}_1$ of order p, the public parameters are perfectly indistinguishable.

KeyGen : The simulator $\mathcal{S}$ chooses a random seed $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function Φ. It computes keys for the regular signature scheme $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.

Then, it chooses $r_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [n]$ and sets $R_{\tau_i} = g_1^{r_i} \cdot (g_1^a)^{\sum_{j=1}^{T} s_j m_{i,\mu}[j]}$. It gives $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, R_{\tau_1}, \ldots R_{\tau_n})$ to the adversary. Since the s_j are chosen uniformly at random this is perfectly indistinguishable form an honestly generated verification key.

Queries: Let l be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $l = 1$). For every new queried dataset Δ, simulator $\mathcal{S}$ creates a list L_Δ of tuples (τ, m), which collects all the label/message pairs queried by the adversary on Δ.

Moreover, whenever the l-th new dataset Δ_l is queried and $l \neq \mu$, $\mathcal{S}$ does the following: On query (τ, m) the simulator $\mathcal{S}$ computes $z \leftarrow \mathcal{R}(\Delta_l)$. as well as $\sigma_{\Delta_l} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta_l || Z)$. Then, it computes $\Lambda \leftarrow (R_\tau \cdot \prod_{j=1}^{T} H_j^{m[j]})^z$ and returns the signature $\sigma = (\sigma_{\Delta_l}, Z, \Lambda)$. Note that these are exactly the same as honestly generated signatures.

If $l = \mu$ $\mathcal{S}$ does the following: On query (τ, m) the simulator $\mathcal{S}$ chooses $u \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $Z = \left(g_2^b\right)^u$, as well as $\sigma_{\Delta_\mu} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta_\mu || Z)$. Then the simulator $\mathcal{S}$ computes $\Lambda = \left(g_1^b\right)^{ur_\tau}$ and returns the signature $\sigma = (\sigma_{\Delta_\mu}, Z, \Lambda)$. Note that we have

$$\Lambda = (g_1^{r_\tau})^{ub} = \left(g_1^{r_\tau - \sum_{j=1}^{T} a s_j m[j] + \sum_{j=1}^{T} a s_j m[j]}\right)^{ub} = \left(R_\tau \cdot \prod_{j=1}^{T} H_j^{m[j]}\right)^z$$

and thus the simulated signatures are perfectly indistinguishable from honestly generated signatures.

Forgery: Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by the adversary $\mathcal{A}$. Parse $\sigma^* = (\sigma^*_{\Delta^*}, Z^*, \Lambda^*))$ and $\mathcal{P}^*_{\Delta^*} = (f*, \tau_1, \ldots, \tau_n, \Delta^*)$. The simulator evaluates the function f^* over the dataset identified by Δ^*, i.e. it computes $\hat{m} =$

$f^*(m_{1,\mu}, ..., m_{n,\mu})$ and $\hat{\sigma} = (\hat{\sigma}_\Delta, \hat{Z}, \hat{\Lambda}) \leftarrow \mathsf{Eval}(\mathsf{ek}, f^*, \sigma_1, \ldots, \sigma_n)$. Note that we have $\prod_{j=1}^{T} H_j^{\hat{m}[j]} \neq \prod_{j=1}^{T} H_j^{m^*[j]}$, since $\mathsf{bad}_5 = \mathsf{false}$ and therefore also $\sum_{j=1}^{T} s_j \cdot \hat{m}[j] \neq \sum_{j=1}^{T} s_j \cdot m^*[j]$.

It returns $(\hat{\Lambda} \cdot (\Lambda^*)^{-1})^{(\sum_{j=1}^{T} s_j(m^*[j]-\hat{m}[j]))^{-1}}$ as a solution. Let $R = \prod_{i=1}^{n} R_{\tau_i}^{f_i}$. Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ by assumption and $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}^*_{\Delta^*}, \hat{m}, \hat{\sigma}) = 1$ due to Propositions 6.14 and 6.15, we have:

$$\hat{\Lambda} = \left(R \cdot \prod_{j=1}^{T} H_j^{\hat{m}[j]} \right)^{ub} = \left(R^{ub} \cdot (g_1^a)^{ub(\sum_{j=1}^{T} s_j \hat{m}[j])} \right) = R^{ub} \cdot g_1^{(\sum_{j=1}^{T} s_j \hat{m}[j])uab}$$

$$\Lambda^* = \left(R \cdot \prod_{j=1}^{T} H_j^{m^*[j]} \right)^{ub} = R^{ub} (g_1^a)^{ub(\sum_{j=1}^{T} s_j m^*[j])} = R^{ub} g_1^{(\sum_{j=1}^{T} s_j m^*[j])uab}$$

Therefore, we have

$$\begin{aligned} \hat{\Lambda} \cdot (\Lambda^*)^{-1} &= (R^{ub} \cdot g_1^{u(\sum_{j=1}^{T} s_j \cdot \hat{m}[j])ab}) \cdot (R^{-ub} \cdot g_1^{-u(\sum_{j=1}^{T} s_j \cdot m^*[j])ab}) \\ &= g_1^{(\sum_{j=1}^{T} s_j \cdot \hat{m}[j])uab} \cdot g_1^{-(\sum_{j=1}^{T} s_j \cdot m^*[j])uab} = g_1^{ab(u \sum_{j=1}^{T} s_j(\hat{m}[j]-m^*[j]))} \end{aligned}$$

which yields

$$(\Lambda \cdot (\Lambda^*)^{-1})^{\frac{1}{u \sum_{j=1}^{T} s_j(\hat{m}[j]-m^*[j])}} = (g_1^{ba})^{\frac{u \sum_{j=1}^{T} s_j(\hat{m}[j]-m^*[j])}{u \sum_{j=1}^{T} s_j(\hat{m}[j]-m^*[j])}} = g_1^{ab}$$

As we have shown above $\sum_{j=1}^{T} s_j(m^*[j] - \hat{m}[j]) \neq 0$ and therefore we have $\Pr[\mathbf{Adv}(\mathcal{S})] = \Pr[G_5(\mathcal{A})]$, which proves the statement.

□

Implementation

We now report on the experimental results of a Rust implementations of Construction 6.13. The measurements are based on an implementation by Rune Fiedler and Lennart Braun. As a pairing group the BLS curve [15] *BLS12-381* [27] is used.

The following measurements were executed on an Intel Core i7-4770K (Haswell) processor running at 3.50 GHz.

We present the runtimes of the individual subalgorithms of linearly homomorphic authenticator scheme presented in Construction 5.3.

We first present the runtimes influenced by the dimension T of vectors $m \in \mathbb{Z}_p^T$ given as messages in Table 6.1. Then, we present the runtimes influenced by the number of inputs n messages in Table 6.2.

Dimension	Setup	Auth	EffVer
32	6128	10563	13971
64	12313	19860	23576
128	24308	38563	42148
256	49046	75955	79691
512	97483	150746	154182

Table 6.1: Runtimes of $\mathsf{CDH} - \mathsf{LinAuth}$ 4.13 in μs

Inputs	KeyGen	Eval	VerPrep
256	49064	76465	74882
512	98237	152511	149454
1024	196331	305536	299514
2048	392655	6106054	598618
4096	784895	12212208	1197337

Table 6.2: Runtimes of $\mathsf{CDH} - \mathsf{LinAuth}$ 4.13 in μs

6.4 CDH Based Linearly Homomorphic Authenticated Encryption

In this section, we show how our linearly homomorphic signature scheme can be used to instantiate such a LAEPuV scheme $\mathsf{LAE} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AEval}, \mathsf{AVerify}, \mathsf{ADecrypt})$ when using bilinear groups of *composite order*. In [25] it is shown how to construct even asymmetric bilinear groups of composite order $n = pq$. Note that previous instantiations of LAEPuV schemes can only sign messages in $\mathbb{Z}_n$, i.e. vectors of length 1, while we show the first use of LAEPuV for vectors of *polynomial length*.

Construction 6.25.

$\mathsf{AKeyGen}(1^\lambda, n, T)$**:** *On input a security parameter λ, an integer n, and an integer T, it chooses two (safe) primes p, q and computes the modulus $N \leftarrow p \cdot q$. It runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group $\mathsf{bgp} = (N, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, e)$ of composite order and samples $n + T$ elements $R_1, \ldots, R_n, h_1, \ldots h_T \xleftarrow{\$} \mathbb{G}_1$ uniformly at random. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ and a a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_N$. The algorithm chooses a random seed $K, K' \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function Φ. It computes keys for the regular signature scheme $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. Furthermore it chooses an element $g \in \mathbb{Z}^*_{N^2}$ of order N as well*

as a hash function $H : \{0,1\}^* \to \mathbb{Z}_{N^2}^*$. *It returns the key pair* $(\mathsf{sk}, \mathsf{pk})$ *with* $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, K, p, q)$ *and* $\mathsf{pk} = (\mathsf{bgp}, H, \mathsf{pk}', g, \{h_j\}_{j=1}^T, \{R_i\}_{i=1}^n)$.

$\mathsf{AEncrypt}(\mathsf{sk}, \Delta, \tau, m)$**:** *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* $\tau \in \mathcal{T}$*, and a message* $m \in \mathbb{Z}_N^T$*, it chooses* $\beta_j \xleftarrow{\$} \mathbb{Z}_{N^2}$ *uniformly at random for* $j \in [T]$*. It computes the ciphertext* $C[j] \leftarrow g^{m[j]} \cdot \beta[j]^N \mod N^2$*, computes* $S[j] \leftarrow H(\Delta||\tau||j)$ *and computes* $(a[j], b[j]) \in \mathbb{Z}_N \times \mathbb{Z}_N^*$ *such that* $g^{a[j]} \cdot b[j]^N = C[j]S[j] \mod N^2$ *using the factorization of* N *(see [84] for a detailed description). It generates the parameters for the dataset identified by* Δ*, by running* $z \leftarrow \Phi_K(\Delta)$ *and computing* $Z = g_2^z$*. It binds* Z *to the dataset identifier* Δ *by using the regular signature scheme, i.e. it sets* $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$*. Then, it computes* $\Lambda \leftarrow (R_\tau \cdot \prod_{j=1}^T h_j^{-a[j]})^z$ *and returns the the ciphertext* $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$.

$\mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^n)$**:** *On input a public key* pk*, a multi-labeled program* $\mathcal{P}_\Delta$*, and a set of ciphertexts* c_i*, it parses* $\mathcal{P}_\Delta = (f, \tau_1, \ldots, \tau_n, \Delta)$ *and* $c_i = (C_i, a_i, b_i, \sigma_{\Delta,i}, Z_i, \Lambda_i)$*. If* $Z_i \neq Z_1$ *for any* $i \in [n]$*, it aborts. Otherwise, it sets*

$$C \leftarrow \prod_{i=1}^n C_i^{f_i} \mod N^2 \qquad a \leftarrow \sum_{i=1}^n f_i a_i \mod N$$

$$b[j] \leftarrow \prod_{i=1}^n b_i[j]^{f_i} \mod N^2, \text{ for } j \in [T] \qquad \Lambda \leftarrow \prod_{i=1}^n \Lambda_i^{f_i} \mod N$$

It returns the ciphertext $c = (C, a, b, \sigma_{\Delta,1}, Z_1, \Lambda)$.

$\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c)$**:** *On input a public key* pk*, a multi-labeled program* $\mathcal{P}_\Delta$*, and a ciphertext* c*, it parses* $\mathcal{P}_\Delta = ((f_1, ..., f_n), \tau_1, \ldots, \tau_n, \Delta)$ *and* $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$*. The algorithm checks whether the following equations hold:* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$ $e\left(R \cdot \prod_{j=1}^T h_j^{-a[j]}, Z\right) = e(\Lambda, g_2)$*, and* $g^{a[j]} \cdot b[j]^n = C[j] \prod_{i=1}^n H(\Delta|\tau_i|j)^{f_i} \mod N^2$*. If all checks are satisfied, it returns* '1'*. Otherwise, it returns* '0'.

$\mathsf{ADecrypt}(\mathsf{sk}, \mathcal{P}_\Delta, c)$**:** *Returns* $\perp$ *if* $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 0$*. Otherwise, it computes* (m, β) *such that* $g^{m[j]}\beta[j]^N = C[j] \mod n^2$ *and return* m.

We will explicitly show this combination of primitives is still correct. The first instantiation of a LAEPuV scheme described in [43] was indeed not correct as it suffered from false positives as shown in [98].

Theorem 6.26. *The LAEPuV scheme* LAE *is correct in the sense of Definition 2.20.*

Proof. We fix a random key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda, n, T)$, with $\mathsf{sk} = (\mathsf{sk}', K, p, q)$ and $\mathsf{pk} = (\mathsf{bgp}, H, \mathsf{pk}', g, \{h_j\}_{j=1}^T, \{R_i\}_{i=1}^n)$.

1. If $g \in \mathbb{Z}_{N^2}^*$ has order N then the map: $\mathbb{Z}_n \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$, $(a,b) \mapsto g^a \cdot b^N$ is an isomorphism (see [84]). If $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, c) = 1$ holds then we have in particular $g^{a[j]} \cdot b[j]^n = C[j] \prod_{i=1}^k H(\Delta||\tau_i||j)^{f_i} \mod N^2$, where each $g^{a[j]} \cdot b[j]^n$ and $H(\Delta||\tau_i||j)^{f_i}$ is an element of $\mathbb{Z}_{N^2}^*$. Since this is a group so is every $C[j]$ which means every Paillier decryption yields a valid message m.
2. We choose messages $m_i \xleftarrow{\$} \mathbb{Z}_N^T$ as well as a dataset identifier $\Delta \in \{0,1\}^*$ and a multi-labeled program $\mathcal{P}_\Delta = ((f_1, \ldots, f_n), \tau_1, \ldots, \tau_n, \Delta)$. Let $c_i \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \Delta, i, m_i)$ and $c \leftarrow \mathsf{AEval}(\mathsf{pk}, \mathcal{P}_\Delta, \{c_i\}_{i=1}^n)$.
   By definition we have $c = (C, a, b, \sigma_\Delta, Z, \Lambda)$. Where for each $j \in [T]$ we have

$$C[j] = \prod_{i=1}^n \left(g^{m_i[j]} \beta_i[j]^N\right)^{f_i} = g^{\sum_{i=1}^n f_i m_i[j]} \left(\prod_{i=1}^n \beta_i[j]^{f_i}\right)^N \mod N^2$$

$$g^{a[j]} \cdot b[j]^N = g^{\sum_{i=1}^n f_i a_i[j]} \cdot \left(\prod_{i=1}^n b_i[j]^{f_i}\right)^N = \prod_{i=1}^n \left(C[j]^{f_i} \cdot H(\Delta||\tau_i||j)^{f_i}\right)$$

$$= C \cdot \prod_{i=1}^n (H(\Delta||\tau_i||j)^{f_i}) \mod N^2$$

$$z = \Phi_K(\Delta),\ Z = g_2^z,\ \sigma_\Delta = \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z).$$

$$\Lambda = \prod_{i=1}^n \Lambda_i^{f_i} = \left(\prod_{i=1}^n R_i^{f_i} \cdot \prod_{j=1}^T h_j^{-\sum_{i=1}^n f_i a_i[j]}\right)^z = \left(R \cdot \prod_{j=1}^T h_j^{-a[j]}\right)^z$$

   Therefore we have $\mathsf{AVerify}(\mathsf{pk}, \mathcal{P}_\Delta, C) = 1$ and due to the first equation Paillier decryption of $C[j]$ yields $\sum_{i=1}^n f_i m_i[j]$ for each $j \in [T]$.

□

We now show the security of Construction 6.25 by applying the generic results of Catalano et al. [43].

Theorem 6.27. *([43]). In the random oracle model, if the DCR Assumption (see Def. 2.44), the DL assumption (see Def. 2.36) the co − DHP* Assumption (see Def. 2.39) hold and H is a random oracle the LAEPuV scheme* LAE *is a LH-IND-CCA secure (see Definition 2.21) LAEPuV scheme.*

Proof. This is a direct corollary of Theorem 2.24 and Theorem 6.19. □

Theorem 6.28. *([43]) Construction 6.25 is LH-Uf-CCA secure according to Definition 2.22.*

Proof. This is a direct corollary of Theorem 2.25 and Theorem 6.19. □

7 Adding Information-Theoretic Privacy to Homomorphic Authenticators

Time-consuming computations are commonly outsourced to the cloud. Such infrastructures attractively offer cost savings and dynamic computing resource allocation. In such a situation, it is desirable to be able to verify the outsourced computation. The verification must be *efficient*, by which we mean that the verification procedure is significantly faster than verified computation itself. Otherwise, the verifier could as well carry out the computation by himself, negating the advantage of outsourcing.

Often, not only the data owner is interested in the correctness of a computation; but also third parties, like insurance companies in the case of medical data. In addition, there are scenarios in which computations are performed over sensitive data. For instance, a cloud server may collect health data of individuals and compute aggregated data. Hence the requirement for efficient verification procedures for outsourced computing that are privacy-preserving, both for computation inputs and for computation results.

Growing amounts of data are sensitive enough to require long-term protection. Electronic health records, voting records, or tax data require protection periods exceeding the lifetime of an individual. Over such a long time, complexity-based confidentiality protection is unsuitable because algorithmic progress is unpredictable. In contrast, *information-theoretic* confidentiality protection is not threatened by algorithmic progress and supports long-term security.

In Chapters 5 and 6 we presented authenticator based solutions that address verifiability and privacy to certain degrees. The schemes in Chapter 5 achieve information-theoretic input privacy with respect to the verifier, while the schemes in Chapter 6 achieve computational input and output privacy with respect to the servers. In Chapter 4 we presented the first solution to achieve complete information-theoretic privacy (input and output privacy with respect to the verifierinput and output privacy with respect to the servers). To this end we used our novel function-

dependent commitment scheme (FDC), which combines the privacy properties of commitments with the verifiability properties of homomorphic authenticators. In this chapter we discuss how to add the privacy properties of FDCs to known homomorphic authenticator schemes in order to makes them suitable even when sensitive data are processed.

Contribution. In this chapter, we investigate the relation between homomorphic authenticators and FDCs. Our contribution is threefold.

First, we show how every FDC can be transformed into a homomorphic authenticator scheme, showing that FDCs are at least as powerful schemes as homomorphic authenticators. To this end, we explicitly construct the algorithms making up a homomorphic authenticator scheme, using only the algorithms from the input FDC. We then derive both authentication correctness and evaluation correctness for the homomorphic authenticator scheme output by our transformation. The proof relies on the correctness of the input FDC scheme. For security, we derive the unforgeability of the resulting homomorphic authenticator scheme from two conditions on the underlying FDC: its own unforgeability, and its bindingness. Regarding bandwidth, we prove that the output homomorphic authenticator scheme is succinct if the input FDC scheme is succinct.

Second, we show how an FDC can be generically constructed from a *structure-preserving* homomorphic authenticator scheme, assuming the additional existence of a homomorphic commitment scheme and of a separate classical commitment scheme. We require the commitment space of the homomorphic commitment scheme to be a subset of the structure preserved by the homomorphic authenticator scheme. The message space of the classical commitment scheme allows labeled programs as admissible inputs, unlike the homomorphic commitment scheme. We show that if the two underlying commitment schemes are binding, then the resulting FDC inherits this bindingness. Furthermore, we prove that the output FDC inherits the unconditional hiding from the underlying homomorphic commitment scheme. The correctness of the output FDC is shown to follow from three assumptions on the input homomorphic authenticator scheme: authentication correctness, evaluation correctness and efficient verification. Regarding security, we prove that unforgeability is also inherited. This is done by showing that a simulator can forward adversary queries in the FDC security experiment to queries in the homomorphic authenticator experiment. The resulting forgery can be used to compute a forgery in the other experiment. For performance, we show that if the input scheme is succinct, respectively efficiently verifiable, then the output FDC is also succinct, respectively has amortized efficiency. Our transformation enables the use of certain existing homomorphic authenticator schemes in particularly privacy-sensitive settings. Applying this transformation enables information-theoretic output privacy.

This allows interested third parties to verify the correct computation of a function without even needing to learn the result.

Finally we apply the latter transformation to two schemes discussed in this Thesis, thereby presenting the first FDC for quadratic functions as well as the first multi-key FDC.

Organization. In this chapter we investigate the relation between homomorphic authenticators and the FDCs introduced in Chapter 4. We show how to transform any FDC into a homomorphic authenticator (Sec. 7.1). Afterwards we show how to construct FDCs from homomorphic commitments and structure-preserving homomorphic authenticators (Sec. 7.2). We then apply the latter transformation to two variants of the schemes described in Chapter 5. In Sec. 7.3 we use this transformation to obtain a multi-key FDC for linear functions. We then too show how to combine this FDC with secret sharing to derive a verifiable computing scheme with complete information-theoretic privacy. In Sec. 7.4 we use the same transformation to obtain an FDC for multivariate polynomials of degree 2. We then show how to combine this FDC with secret sharing to derive a verifiable computing scheme with complete information-theoretic privacy.

Publications. This chapter is based on publication [S7].

Related Work. Catalano et al. [43] showed a transformation for linearly homomorphic signatures that adds computational privacy. This is described in Chapter 6. In this chapter, however, we address a transformation that adds information-theoretic privacy properties. Libert et al. [78] already presented a structure-preserving linearly homomorphic signature scheme. In this chapter further structure-preserving homomorphic authenticator schemes are presented. The first supports linear functions over data authenticated by multiple keys, the second quadratic functions.

7.1 Homomorphic Authenticators from FDCs

In this section, we begin to investigate the relation between FDCs and homomorphic authenticators. In particular we show how any correct FDC can be transformed into a homomorphic signature scheme and show how it inherits properties from the underlying FDC.

We begin by describing a transformation that constructs the algorithms of a homomorphic authenticator scheme from the algorithms of an FDC.

We describe a transformation Φ, that on input an FDC scheme FDC = (Setup, KeyGen, PublicCommit, PrivateCommit, FunctionCommit, Eval, FunctionVerify,

PublicDecommit) outputs a homomorphic authenticator scheme HAuth = (HSetup, HKeyGen, Auth, HEval, Ver).

Construction 7.1.

- $\mathsf{HSetup}(1^\lambda)$: *On input a security parameter* λ*, this algorithm runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*. It outputs the public parameters* pp*.*
- $\mathsf{HKeyGen}(\mathsf{pp})$: *On input the public parameters* pp *it runs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*. It outputs the secret key* sk*, the evaluation key* $\mathsf{ek} = 0$ *as well as the verification key* $\mathsf{vk} = \mathsf{pk}$*.*
- $\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$: *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* τ*, and a message* m*, it chooses randomness* $r \xleftarrow{\$} \mathcal{R}$ *uniformly at random. It runs* $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$*. It sets* $M = m$ *and outputs the authenticator* $\sigma = (M, r, A)$*.*
- $\mathsf{HEval}(f, \{\sigma_i\}_{i\in[n]}, \mathsf{ek})$: *On input an function* $f : \mathcal{M}^n \to \mathcal{M}$*, a set* $\{\sigma_i\}_{i\in[n]}$ *of authenticators, and an evaluation key* ek *(in our construction, no evaluation key is needed), the algorithm parses* $\sigma_i = (M_i, r_i, A_i)$*. It runs* $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$*, and sets* $M^* = f(M_1, \ldots, M_n)$ *as well as* $r^* = \hat{f}(m_1, \ldots, m_n, r_1, \ldots, r_n)$*, where* $\hat{f} \in \mathcal{F}$ *can be derived from* f *if the FDC scheme is correct in the sense of Def. 4.7. It sets* $\sigma = (M^*, r^*, A^*)$ *and outputs* σ*.*
- $\mathsf{Ver}(\mathcal{P}_\Delta, \mathsf{vk}, m, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$*, a verification key* vk*, a message* $m \in \mathcal{M}$*, and an authenticator* σ *, the algorithm parses* $\sigma = (M, r, A)$ *and* $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$*. It checks if* $m = M$*. If not it outputs '0', else it runs* $C \leftarrow \mathsf{PublicCommit}(m, r)$ *as well as* $F \leftarrow \mathsf{FunctionCommit}(\mathsf{vk}, \mathcal{P})$*. It runs* $b \leftarrow \mathsf{FunctionVerify}(\mathsf{vk}, A, C, F, \Delta)$*. It outputs* b*.*

This shows how to formally transform any correct FDC scheme into a homomorphic authenticator scheme. We will now show how the properties of such a homomorphic authenticator scheme are derived from those of the FDC scheme.

For homomorphic authenticators, correctness comes in two flavors. Both authenticators created directly with a secret signing key as well as those derived by the homomorphic property should verify correctly. Both properties are derived from the FDC's correctness.

Proposition 7.2. *If* FDC *satisfies correctness in the sense of Def. 4.7, then Construction 7.1 achieves authentication correctness in the sense of Def. 2.5)and evaluation correctness in the sense of Def. 2.6.*

Proof. Let $\tau \in \mathcal{T}$ be an arbitrary label and $\Delta \in \{0,1\}^*$ be an arbitrary dataset identifier. We consider the multi-labeled identity program $\mathcal{P}_\Delta = \mathcal{I}_{\tau,\Delta}$. Let $m \in \mathcal{M}$ be an arbitrary message, and $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. By construction, we know that $\sigma = (M, r, A)$,with $M = m$, where $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$.

By correctness of the FDC, we know that $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta) = 1$ for $C \leftarrow \mathsf{PublicCommit}(m, r)$ and $F \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$. Now we have $\mathsf{Ver}(\mathcal{I}_{\tau,\Delta}, \mathsf{vk}, m, \sigma) = 1$, and HAuth achieves authentication correctness.

Let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier and $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[N]}$ any set of program/message/authenticator triples such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, m_i, \sigma_i) = 1$ for all $i \in [N]$. Let $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{HEval}(g, \{\sigma_i\}_{i\in[N]}, \mathsf{ek})$. We parse $\sigma_i = (M_i, r_i, A_i)$ and $\sigma^* = (M^*, r^*, A^*)$. We set $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$ and $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$. By construction, we know that $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$, as well as $M^* = f(M_1, \ldots, M_N)$, $r^* = \hat{g}(m_1, \ldots, m_N, r_1, \ldots, r_N)$, where $\hat{g} \in \mathcal{F}$ can be derived from g since the FDC scheme is correct in the sense of Def. 4.7. We set $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$ and $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$. By assumption, $m_i = M_i$ hence $M^* = m^*$. Since FDC satisfies correctness, we know that $\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F^*, \Delta) = 1$ and thus $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, m^*, \sigma^*) = 1$. □

Next, we investigate the relation between the unforgeability notions of FDCs and homomorphic authenticators. We can show that an FDC that is both unforgeable and binding can be transformed into an unforgeable homomorphic authenticator.

Proposition 7.3. *If* FDC *is unforgeable in the sense of Def. 4.10 and binding in the sense of Def. 4.3, then Construction 7.1 is unforgeable in the sense of Def. 2.11.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ (see Def. 2.9), we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to win the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$ (see Def. 4.9).

$\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathsf{FDC}}$:

Setup: Simulator $\mathcal{S}$ receives the public parameters pp from the challenger of the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$. It gives pp to the adversary $\mathcal{A}$.

Key Generation: Simulator $\mathcal{S}$ receives pk from the challenger of the experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$. It sets $\mathsf{ek} = 0$, $\mathsf{vk} = \mathsf{pk}$, and outputs $(\mathsf{ek}, \mathsf{vk})$ to the adversary $\mathcal{A}$.

By construction, $\mathsf{ek} = 0$.

Queries: When $\mathcal{A}$ asks queries (Δ, τ, m) $\mathcal{S}$ chooses $r \in \mathcal{R}$ uniformly at random and queries (Δ, τ, m, r) to receive an authenticator A. It sends the authenticator $\sigma = (m, r, A)$ to $\mathcal{A}$. Note that σ is perfectly indistinguishable from a response to a query (Δ, τ, m) during $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$.

Forgery: $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. $\mathcal{S}$ parses $\sigma^* = (M^*, r^*, A^*)$. It computes the correct results $\hat{m} = f^*(m_1, \ldots, m_n), \hat{r} = \hat{f}^*(m_1, \ldots, m_n, r_1, \ldots, r_n)$. It computes $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$ and then checks whether $C^* = \mathsf{PublicCommit}(\hat{m}, \hat{r})$. If not, it returns $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$

We either find a collision (m^*, r^*) and $(\hat{m}, \hat{r})$ for the bindingness, or produce a forgery since a type 1, 2, 3 forgery in experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$ corresponds exactly to a type 1, 2, 3 forgery in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$.
Therefore we have $\Pr[\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}(\lambda) = 1] \leq \Pr\left[\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda) = 1\right]$
$+ \Pr\left[\mathbf{EXP}^{Bind}_{\mathcal{A},\mathrm{Com}}(\lambda) = 1\right]$. □

We now analyze this transformation with respect to its efficiency. A trivial construction of a homomorphic authenticator scheme is to (conventionally) sign every input, and during HEval to simply concatenate all authenticators along with the corresponding values. Verification then consists of checking every input value, and then redoing the computation. This naive solution is obviously undesirable in terms of bandwidth, efficiency and fails to provide privacy guarantees.

Succinctness guarantees that a homomorphically derived authenticator is still small, thus keeping bandwidth requirements low.

We show that the homomorphic authenticators derived by our transformation inherits this property from the underlying FDC.

Proposition 7.4. *If* FDC *is succinct in the sense of Def. 4.11, then Construction 7.1 is succinct in the sense of Def. 2.7.*

Proof. By assumption, the size of the output of $\mathsf{PrivateCommit}$ and Eval depends at most logarithmically on n. By construction, the size of authenticators thus depends at most logarithmically on n. □

7.2 From Structure-Preserving Homomorphic Authenticators to FDCs

In this section, we discuss how to construct an FDC from (homomorphic) commitment schemes and structure-preserving homomorphic signatures schemes over the commitment space. We show how the properties of the resulting FDC depend on the underlying homomorphic signature scheme and commitment scheme. Note that we do not require all the properties of structure-preserving authenticators. In fact, our transformation can be to any homomorphic commitment scheme with commitment space contained within a homomorphic authenticator scheme, which is homomorphic with respect to the operations in the commitment space.

Assume the homomorphic authenticator scheme $\mathsf{HAuth} = (\mathsf{HSetup}, \mathsf{HKeyGen}, \mathsf{Auth}, \mathsf{HEval}, \mathsf{Ver})$ is structure-preserving over some structure $\mathcal{X}$. Let Com be a homomorphic commitment scheme $\mathsf{Com} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit}, \mathsf{CEval})$ with message space $\mathcal{M}$ and commitment space $\mathcal{C} \subset \mathcal{X}$. We also assume the existence of an ordinary commitment scheme $\mathsf{Com}' = (\mathsf{CSetup}', \mathsf{Commit}', \mathsf{Decommit}')$ with

message space $\mathcal{F} \times \mathcal{T}^n$, so labeled programs are admissible inputs. One can always split up Ver into ($\mathsf{VerPrep}$, EffVer) as follows.

$\mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$: On input a labeled program $\mathcal{P}$ and a verification key vk, the algorithm sets $\mathsf{vk}_{\mathcal{P}} = (\mathcal{P}, \mathsf{vk})$. It returns $\mathsf{vk}_{\mathcal{P}}$.

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, C, \sigma, \Delta)$**:** On input a concise verification key $\mathsf{vk}_{\mathcal{P}}$, a message C, an authenticator σ, and a dataset identifier $\Delta \in \{0,1\}^*$, the algorithm parses $\mathsf{vk}_{\mathcal{P}} = (\mathcal{P}, \mathsf{vk})$. It runs $b \leftarrow \mathsf{Ver}(\mathcal{P}, \mathsf{vk}, C, \sigma, \Delta)$ and returns b.

We now show how to construct an FDC.

Construction 7.5.

$\mathsf{Setup}(1^\lambda)$ *takes the security parameter* λ *as input. It runs* $\mathsf{CK} \leftarrow \mathsf{CSetup}(1^\lambda)$, $\mathsf{CK}' \leftarrow \mathsf{CSetup}'(1^\lambda)$ *as well as* $\mathsf{pp}' \leftarrow \mathsf{HSetup}(1^\lambda)$. *It sets* $\mathsf{pp} = (\mathsf{CK}, \mathsf{CK}', \mathsf{pp}')$ *and outputs* pp. *We implicitly assume that every algorithm uses these public parameters* pp*, leaving them out of the notation.*

$\mathsf{KeyGen}(\mathsf{pp})$ *takes the public parameters* pp *as input and runs* $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{HKeyGen}(\mathsf{pp})$. *It sets* $\mathsf{sk} = (\mathsf{sk}', \mathsf{ek})$, $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ *and outputs the secret-public key pair* $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{PublicCommit}(m, r)$ *takes as input a message* m *and randomness* r *and runs* $(C, d) \leftarrow \mathsf{Commit}(m, r)$. *It outputs the commitment* C.

$\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$ *takes as input the secret key* sk*, a message* m*, randomness* r*, an identifier* τ *and a dataset identifier* Δ. *It runs* $(C, d) \leftarrow \mathsf{Commit}(m, r)$, $A' \leftarrow \mathsf{Auth}(\mathsf{sk}, \tau, \Delta, C)$ *and outputs* $A = (A', \mathsf{ek})$.

$\mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P})$ *takes as input the public key* pk *and a labeled program* $\mathcal{P}$. *It parses* $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ *and runs* $\mathsf{vk}_{\mathcal{P}} \leftarrow \mathsf{VerPrep}(\mathcal{P}, \mathsf{vk})$. *It chooses randomness* $r_{\mathcal{P}} \xleftarrow{\$} \mathcal{R}$ *uniformly at random and runs* $(C_{\mathcal{P}}, d_{\mathcal{P}}) \leftarrow \mathsf{Commit}'(\mathcal{P}, r_{\mathcal{P}})$. *It outputs the function commitment* $F = (\mathsf{vk}_{\mathcal{P}}, C_{\mathcal{P}})$.

$\mathsf{Eval}(f, A_1, \ldots A_n)$ *takes as input a function* f *and a set of authenticators* $A_1, \ldots, A_n$. *It parses* $A_i = (A_i', \mathsf{ek}_i)$ *for all* $i \in [n]$. *Afterwards, it runs* $\hat{A} \leftarrow \mathsf{HEval}(f, \{A_i'\}_{i \in [n]}, \mathsf{ek}_1)$. *It outputs* $A^* = (\hat{A}, \mathsf{ek}_1)$.

$\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F, \Delta)$ *takes as input a public key* pk*, an FDC containing an authenticator* A *and a commitment* C*, a function commitment* F *as well as a dataset identifier* Δ. *It parses* $F = (\mathsf{vk}_{\mathcal{P}}, C_{\mathcal{P}})$*, and* $A = (A', \mathsf{ek})$. *It runs* $b \leftarrow \mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, C, A', \Delta)$ *and outputs* b.

$\mathsf{PublicDecommit}(m, r, C)$ *takes as input message* m*, randomness* r*, and commitment* C. *It runs* $(C, d) \leftarrow \mathsf{Commit}(m, r)$ *as well as* $b \leftarrow \mathsf{Decommit}(m, d, C)$ *and outputs* b.

This shows how to formally obtain an FDC scheme from a structure-preserving HA scheme and a commitment scheme. We will now show how the properties of such an FDC scheme are derived from those of underlying primitives.

We first look at the commitment properties — hiding and binding. In our transformation, these are inherited from the underlying commitment schemes.

Proposition 7.6. *Construction 7.5 is binding in the sense of Def. 4.3 if* Com *and* Com' *used in the construction are binding commitment schemes.*

Proof. Obviously, if Com is binding then $\mathsf{PublicCommit}$ is binding. We parse a function commitment as $F = (\mathsf{vk}_{\mathcal{P}}, C_{\mathcal{P}})$. Note that $C_{\mathcal{P}}$ is by assumption a binding commitment, thus $\mathsf{FunctionCommit}$ is also binding. □

Proposition 7.7. *Construction 7.5 is target binding in the sense of Def. 4.5 if* Com *and* Com' *used in the construction are target binding commitment schemes.*

Proof. Obviously, if Com is target binding then $\mathsf{PublicCommit}$ is target binding. We parse a function commitment as $F = (\mathsf{vk}_{\mathcal{P}}, C_{\mathcal{P}})$. Note that $C_{\mathcal{P}}$ is by assumption a target binding commitment, thus $\mathsf{FunctionCommit}$ is also target binding. □

The hiding property of FDCs (see Def. 4.6) is different from the context hiding property of homomorphic authenticators (see Def. 2.18). On a high level, the context hiding property guarantees that authenticators to the *output* of a computation do not leak information about the *inputs* to the computation. In contrast, the hiding property of FDCs guarantees that even the authenticators to the inputs do not leak information about the inputs to the computation. In [90] this property was used to combine an FDC with secret sharing to construct an efficient verifiable multi-party computation scheme. This gain in privacy in one of the major benefits of FDCs over homomorphic authenticators in cases of sensitive data used as inputs to a computation.

Proposition 7.8. *If* Com *is (unconditionally) hiding, then Construction 7.5 is (unconditionally) hiding in the sense of Def. 4.6.*

Proof. If Com is (unconditionally) hiding, then the probabilistic distributions over the sets $\{\mathsf{Commit}(m, r) \mid r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{Commit}(m', r') \mid r' \xleftarrow{\$} \mathcal{R}\}$ are perfectly indistinguishable for all $m, m' \in \mathcal{M}$. This is independent of any $\tau \in \mathcal{T}$ and any $\Delta \in \{0,1\}^*$. Hence the probabilistic distributions over sets $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C) \mid C \leftarrow \mathsf{Commit}(m, r), r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C') \mid C' \leftarrow \mathsf{Commit}(m', r'), r' \xleftarrow{\$} \mathcal{R}\}$ are perfectly indistinguishable for all $m, m' \in \mathcal{M}, \tau \in \mathcal{T}, \Delta \in \{0,1\}^*$. Since $\{\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C) \mid C \leftarrow \mathsf{Commit}(m, r), r \xleftarrow{\$} \mathcal{R}\} = \{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\}$ for all $m \in \mathcal{M}, \tau \in \mathcal{T}, \Delta \in \{0,1\}^*$ the probabilistic distributions over $\{\mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau) \mid r \xleftarrow{\$} \mathcal{R}\}$ and $\{\mathsf{PrivateCommit}(\mathsf{sk}, m', r', \Delta, \tau) \mid r' \xleftarrow{\$} \mathcal{R}\}$ are also (perfectly) indistinguishable. The case of $\mathsf{PublicCommit}$ is trivial. □

Next, we investigate the homomorphic property of such an FDC. We can show that if the homomorphic authenticator scheme HAuth satisfies both correcntess properties — authentication and evaluation, and furthermore supports efficient verification, then the transformed FDC is also correct.

Proposition 7.9. *If* HAuth *achieves authentication (see Def. 2.5), evaluation correctness (see Def. 2.6), and efficient verification (see Def. 2.8), then Construction 7.5 is correct in the sense of Def. 4.7 with overwhelming probability.*

Proof. Let λ be any security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and let $\Delta \in \{0,1\}^*$ be an arbitrary dataset identifier. Let $m \in \mathcal{M}$ be an arbitrary message and $r \in \mathcal{R}$ arbitrary randomness.
We set $A \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m, r, \Delta, \tau)$, $C \leftarrow \mathsf{PublicCommit}(m, r)$, as well as $F_\mathcal{I} \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{I}_\tau)$, where $\mathcal{I}_\tau$ is the labeled identity program. Then we have $A = \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, C)$. By the authentication correctness of HAuth, we know that $\mathsf{Ver}(\mathcal{I}_{\tau,\Delta}, \mathsf{vk}, C, \sigma) = 1$. Since HAuth achieves efficient verification, $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{I}_\tau}, C, \sigma, \Delta) = 1$ with overwhelming probability. By construction, $\mathsf{FunctionVerify}(\mathsf{pk}, A, C, F_\mathcal{I}, \Delta) = 1$.

Let $\{m_i, \sigma_i, \mathcal{P}_i)\}_{i\in[N]}$ be any set of tuples (parsed as $\sigma_i = (r_i, A_i)$) such that for $C_i \leftarrow \mathsf{PublicCommit}(m_i, r_i)$, $F_i \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}_i)$, $\mathsf{FunctionVerify}(\mathsf{pk}, A_i, C_i, F_i, \Delta) = 1$. This implies $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}_i}, C_i, \sigma_i, \Delta) = 1$, thus $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, C_i, \sigma_i) = 1$ with overwhelming probability.
Then let $m^* = g(m_1, \ldots, m_N)$, $r^* = \hat{g}(m_1, \ldots, m_N, r_1, \ldots r_N)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$, $F^* \leftarrow \mathsf{FunctionCommit}(\mathsf{pk}, \mathcal{P}^*)$, $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_N)$, and $\sigma^* = (r^*, A^*)$. From the homomorphic property of Com, we have $C^* = \mathsf{CEval}(g, C_1, \ldots, C_N)$. By the evaluation correctness of HAuth we have $\mathsf{Ver}(\mathcal{P}^*, \mathsf{vk}, C^*, \sigma^*) = 1$. Thus $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}^*}, C^*, \sigma^*, \Delta) = 1$ with overwhelming probability, due to the correctness property of efficient verification (see Def. 2.8). By construction, $\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F^*, \Delta) = 1$. □

We now look at the essential security property of an FDC — unforgeability. We show how an adversary that can break the security experiment for FDCs can be used to break the security experiment for homomorphic authenticators. On a high level, we show that a simulator can forward the queries used by the adversary in the FDC experiment, as queries in the homomorphic authenticator experiment and uses the resulting forgery in the one experiment to compute a forgery in the other.

Proposition 7.10. *If* HAuth *is unforgeable in the sense of Def. 2.11, then Construction 7.5 is unforgeable in the sense of Def. 4.10.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$, we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to win the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$.

Setup: Simulator $\mathcal{S}$ receives the public parameters pp' from the challenger of the experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. It runs $\mathsf{CK} \leftarrow \mathsf{CSetup}(1^\lambda)$, $\mathsf{CK}' \leftarrow \mathsf{CSetup}'(1^\lambda)$. It sets $\mathsf{pp} = (\mathsf{CK}, \mathsf{CK}', \mathsf{pp}')$ and outputs pp to the adversary $\mathcal{A}$.

Key Generation: Simulator $\mathcal{S}$ receives $(\mathsf{ek}, \mathsf{vk})$ from the challenger of the experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. It sets $\mathsf{pk} = (\mathsf{ek}, \mathsf{vk})$ and outputs pk to the adversary $\mathcal{A}$.

Queries: When $\mathcal{A}$ ask queries (Δ, τ, m, r), $\mathcal{S}$ computes $(C, d) \leftarrow \mathsf{Commit}(m, r)$ and queries (Δ, τ, C) to receive an authenticator σ. It sets $A = \sigma$ and replies to the query with the private commitment A. This is the exact same reply to a query in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$.

Forgery: The adversary $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, r^*, A^*)$. $\mathcal{S}$ computes $(C^*, d^*) \leftarrow \mathsf{Commit}(m^*, r^*)$ and outputs $(\mathcal{P}^*_{\Delta^*}, C^*, A^*)$.

A type 1, 2, 3 forgery in experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$ corresponds to a type 1, 2, 3 forgery in experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{HAuth}}$. Thus $\mathcal{S}$ produces a forgery with the same probability as $\mathcal{A}$. □

We now analyze an FDC obtained by our transformation with respect to its efficiency properties. On the one hand we have succinctness, which guarantees that authenticators are short, so bandwidth requirements are low. On the other hand, we show how the FDC inherits amortized efficiency, i.e. efficient verification after a one time preprocessing from the efficient verification of the underlying homomorphic authenticator scheme.

Proposition 7.11. *If* HAuth *is succinct in the sense of Def. 2.7, then Construction 7.5 is succinct in the sense of Def. 4.11.*

Proof. By assumption, HAuth produces authenticators whose size depends at most logarithmically on the data set size n. By construction, the output size of PrivateCommit and Eval thus depends at most logarithmically on n. □

Proposition 7.12. *If* HAuth *is efficiently verifiable in the sense of Def. 2.8, then Construction 7.5 achieves amortized efficiency in the sense of Def. 4.12.*

Proof. Let $t(n)$ be the runtime of $f(m_1, \ldots, m_n)$. We recall that FunctionVerify parses a function commitment $F = (\mathsf{vk}_\mathcal{P}, C_\mathcal{P})$ and runs $\mathsf{EffVer}(\mathsf{vk}_\mathcal{P}, C, A, \Delta)$. By assumption, the runtime of EffVer is $o(t(n))$. Thus the runtime of FunctionVerify is also $o(t(n))$. □

7.3 A Multi-Key FDC

We now seek to apply the transformation described in Section 7.2 (Construction 7.5) to a further suitable homomorphic authenticator scheme. We recall that in Chapter 5

we presented a multi-key homomorphic authenticator scheme (Construction 5.3). In the following, we present a slight variation of that scheme that, together with a a generalized Pedersen commitment scheme, will be used in our transformation. We recall that Construction 5.3 on its own provided information-theoretic input privacy with respect to the verifier. By applying our transformation and combining the resulting FDC with a suitable secret sharing scheme we obtain a verifiable computing scheme that provides both information-theoretic input and output privacy with respect to the verifier and information-theoretic input and output privacy with respect to the servers. This allows for use cases where this kind of complete privacy is required.

7.3.1 A Structure Preserving Multi-Key Linearly Homomorphic Authenticator Scheme

We will now showcase a structure preserving variant of our construction MKLin (see Construction) 5.3). More precisely, we will provide a homomorphic authenticator very similar to Construction 5.3. The main difference is the domain of the inputs to Auth. In MKLin messages were elements of $\mathbb{Z}_p^T$. In this variant, messages are taken from $\mathbb{G}_1$. In the original MKLin the component A is computed using $\prod_{j=1}^{T} H_j m[j]$. Note that we have $(H_0^0 \cdot \prod_{j=1}^{T} H_j m[j], 0) \leftarrow \mathsf{Commit}(m, 0)$. So the original MKLin can be seen as taking non-randomized (and therefore not hiding) commitments as inputs. In the following we will detail the more general case of randomized commitments, $\mathsf{Commit}(m, r)$, where $r \xleftarrow{\$} \mathbb{Z}_p$ is taken uniformly at random. The second difference to MKLin is the lack of an S component which we used to make the scheme context hiding. Due to the randomization of the commitment this is not required in this variation.

Construction 7.13 ($\mathsf{SP} - \mathsf{MKLin}$)**.**

$\mathsf{Setup}(1^\lambda)$: *On input a security parameter λ, this algorithm chooses the parameters $k, n, T \in \mathbb{Z}$, a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. It sets the message space $\mathcal{M} = \mathbb{G}_1$. Then it chooses the tag space $\mathcal{T} = [n]$, and the identity space $\mathsf{ID} \subset \{0,1\}^*$. Additionally it fixes a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$, as well as a conventional signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ with signature space Σ contained in the bilinear group. More precisely we have $\Sigma \subset \mathbb{G}_1^{l_1} \times \mathbb{G}_2^{l_2} \times \mathbb{G}_T^{l_T}$ for some $(l_1, l_2, l_T) \in \mathbb{N}_0^3$. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{ID}, \mathsf{bgp}, \Phi, \mathsf{Sig}, \lambda)$.*

$\mathsf{KeyGen}(\mathsf{pp})$: *On input the public parameters pp, the algorithm chooses $K \xleftarrow{\$} \mathcal{K}$ uniformly at random. It runs $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It chooses $x_1, \ldots, x_n, y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It sets $h_i = g_t^{x_i}$ for all $i \in [n]$,*

as well as $Y = g_2^y$. *It sets* $\mathsf{sk} = (K, \mathsf{sk}_{\mathsf{Sig}}, x_1, \dots x_n, y)$, $\mathsf{ek} = \varnothing$, $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_1, \dots, h_n, Y)$ *and outputs* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$. *Each client has its own identity* id *and performs* $\mathsf{KeyGen}(\mathsf{pp})$ *individually, and hence obtains its own key tuple* $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$.

$\mathsf{Auth}(\mathsf{sk}, \Delta, l, M)$: *On input a secret key* sk, *a dataset identifier* Δ, *a label* $l = (\mathsf{id}, \tau)$, *and a message* $M \in \mathbb{G}_1$, *the algorithm computes* $z = \Phi_K(\Delta)$, *sets* $Z_{\mathsf{id}} = g_2^z$ *and binds this parameter to the dataset by signing it, i.e. it computes* $\sigma_{\Delta,\mathsf{id}} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z_{\mathsf{id}}||\Delta)$. *Then it chooses* $r \in \mathbb{Z}_p$ *uniformly at random and sets* $R = g_1^r$. *It parses* $l = (\mathsf{id}, \tau)$ *and computes* $A_{\mathsf{id}} = \left(g_1^{x_l + r} \cdot M\right)^{\frac{1}{z}}$, *where* x_l *is the* x_τ *created during* KeyGen *by identity* id, *and* $C_{\mathsf{id}} = M^{\frac{1}{y}}$. *It sets* $\Lambda = \{(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}})\}$ *and outputs* $\sigma = (\Lambda, R)$.

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]})$: *On input an function* $f : \mathcal{M}^n \to \mathcal{M}$ *and a set* $\{(\sigma_i, \mathsf{eks}_i)\}_{i\in[n]}$ *of authenticators and evaluation keys (in our construction, no evaluation keys are needed, so this set contains only authenticators), the algorithm parses* $f = (f_1, \dots f_n)$ *as a coefficient vector. It parses each* σ_i *as* (Λ_i, R_i) *and sets* $R = \prod_{i=1}^n R_i^{f_i}$. *It sets* $L_{\mathsf{ID}} = \bigcup_{i=1}^n \{\mathsf{id}_i\}$. *It chooses* $(\sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}) \xleftarrow{\$} \{(\sigma, Z) \mid \exists\ A, C \mid (\mathsf{id}, \sigma_\Delta, Z, A, C) \in \bigcup_{i=1}^n \Lambda_i\}$. *Then it computes* $A_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i = \mathsf{id}}}^{n} A_i^{f_i}$, $C_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i = \mathsf{id}}}^{n} C_i^{f_i}$, *and sets* $\Lambda_{\mathsf{id}} = \{\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}}\}$. *Set* $\Lambda = \bigcup_{\mathsf{id}\in L_{\mathsf{ID}}} \Lambda_{\mathsf{id}}$. *It returns* $\sigma = (\Lambda, R)$.

$\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, M, \sigma)$: *On input a multi-labeled program* $\mathcal{P}_\Delta$, *a set of verification key* $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$, *corresponding to the identities* id *involved in the program* $\mathcal{P}$, *a message* $M \in \mathbb{G}_1$, *and an authenticator* σ , *the algorithm parses* $\sigma = (\Lambda, R)$. *For each* id *such that* $(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}}) \in \Lambda$ *it takes* $\mathsf{pk}_{\mathsf{Sig,id}}$ *from* $\mathsf{vk}_{\mathsf{id}}$ *and checks whether* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ *holds, i.e. whether* $\sigma_{\Delta,\mathsf{id}}$ *is a valid signature on* $(Z_{\mathsf{id}}||\Delta)$. *If any check fails it returns '0'. Otherwise it checks whether the following equations hold:* $\prod_{\mathsf{id}\in\mathcal{P}} e(A_{\mathsf{id}}, Z_{\mathsf{id}}) = \prod_{i=1}^n h_{l_i}^{f_i} \cdot \prod_{\mathsf{id}\in\mathcal{P}} e(C_{\mathsf{id}}, Y_{\mathsf{id}}) \cdot e(R, g_2)$, *as well as* $e(\prod_{\mathsf{id}\in\mathcal{P}} C_{\mathsf{id}}, g_2) = e(M, g_2)$. *If they do, it outputs '1', otherwise it outputs '0'.*

Theorem 7.14. *The construction 7.13 is linearly homomorphic and structure preserving over the structure* bgp *if* Sig *has a message space* Σ *for which the following holds: There exists* $(l_1, l_2, l_T) \in \mathbb{N}_0^3$ *such that* $\Sigma \subset \mathbb{G}_1^{l_1} \times \mathbb{G}_2^{l_2} \times \mathbb{G}_T^{l_T}$.

Proof. First we see that message are elements of $\mathbb{G}_1$. Signature components lie either in $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$ if the signature components produced by Sig lie in $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$.

Second we show that authentication correctness in the sense of Def. 2.5 holds for construction 7.13 as well.

To that end we let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple,

$l \in \mathsf{ID} \times \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $M \in \mathbb{G}_1$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, l, M)$. We parse $\sigma = (\Lambda, R)$ and $\Lambda = (\mathsf{id}, \sigma_\Delta, Z, A, C)$.

By construction we have $\sigma_{\Delta,\mathsf{id}} \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, Z_{\mathsf{id}}||\Delta)$ and if Sig is a correct signature scheme then $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig},\mathsf{id}}, Z_{\mathsf{id}}||\Delta, \sigma_\Delta) = 1$ holds. We have by construction

$$\begin{aligned} e(A,Z) &= e\left(\left(g_1^{x_l+r} \cdot M\right)^{\frac{1}{z}}, g_2^z\right) = e\left(g_1^{x_l} \cdot M, g_2\right) \\ &= g_t^{x_l+r} \cdot e\left(M^{\frac{1}{y}m[j]}, g_2^y\right) = h_l \cdot e(C,Y) \cdot e(R,g_2) \end{aligned}$$

as well as $e(C,Y) = e\left(M^{\frac{1}{y}}, g_2^y\right) = e(M,g_2)$, and thus $\mathsf{Ver}(\mathcal{I}_{l,\Delta}, \mathsf{vk}, M, \sigma) = 1$ holds.

Third we show that evaluation correctness in the sense of Def. 2.6 holds for construction 7.13 as well. To that end we let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id}\in\mathsf{ID}}$ be a set of arbitrary key triples, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier. Let $g : \mathcal{M}^N \to \mathcal{M}$ be an arbitrary linear function given by its coefficient vector $(g_1, \ldots, g_N)$. Let $\{(\mathcal{P}_i, M_i, \sigma_i)\}_{i\in[N]}$ be an arbitrary set of program/message/authenticator triples, such that $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, M_i, \sigma_i) = 1$.

Let $M^* = g(M_1, \ldots, M_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$, and $\sigma^* = \mathsf{Eval}(g, \{(\sigma_i, \mathsf{ek}_i)\}_{i\in[n]})$.

Since we have $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, M_i, \sigma_i) = 1$ for all $i \in [n]$, in particular $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig},\mathsf{id}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ holds for all $\mathsf{id} \in \mathcal{P}_i$.

Like in Proposition 5.5 we make use of the following notation: We write $\mathsf{id} \in \mathcal{P}$ if for $\mathcal{P} = (f, l_1, \ldots, l_n)$ there exists an $i \in [n]$ such that $l_i = (\mathsf{id}, \tau_i)$ for some input identifier τ_i.

We have $\bigcup_{i=1}^N \{\mathsf{id} \in \mathcal{P}_i\} = \{\mathsf{id} \in \mathcal{P}^*\}$. Therefore we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}_{\mathsf{id}}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ for all $\mathsf{id} \in \mathcal{P}^*$. We parse $\sigma_i = (\Lambda_i, R_i)$.

If $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \mathsf{vk}, M_i, \sigma_i) = 1$, holds, then in particular

$$\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e(A_{\mathsf{id},i}, Z_{\mathsf{id}}) = \prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} C_{\mathsf{id},i}, Y_{\mathsf{id}}\right) \cdot e(R_i, g_2)$$

holds as well as $e\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} C_{\mathsf{id},i}, g_2\right) = e(M, g_2)$ for all $i \in [N]$. Without loss of generality let $\{\mathsf{id} \in \mathcal{P}_{i,\Delta}\} = \{\mathsf{id} \in \mathcal{P}_{j,\Delta}\}$ for all $i, j \in [n]$. Let f_k for $k \in [n]$ denote the coefficients such that for $\mathcal{P} = (f, l_1, \ldots, l_n) = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$ we have $f(m_1, \ldots, m_n) = \sum_{k=1}^n f_k m_m$. Then we have $f_k = \sum_{i=1}^N g_i f_{i,k}$. We have

$$\prod_{i=1}^{N} \left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e(A_{\mathsf{id},i}, Z_{\mathsf{id},i})\right)^{g_i} = \prod_{i=1}^{N} \left(\prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot \prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e(C_{\mathsf{id},i}, Y_{\mathsf{id}}) \cdot e(R_i, g_2)\right)^{g_i}$$

and

$$\prod_{\mathsf{id}\in\mathcal{P}^*_\Delta} e\left(A^*_{\mathsf{id}}, Z^*_{\mathsf{id}}\right) = \prod_{i=1}^{N}\left(\prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(A_{\mathsf{id},i}, Z_{\mathsf{id},i}\right)\right)^{g_i}$$
$$= \prod_{i=1}^{N}\left(\prod_{k=1}^{n} h_{l_{i,k}}^{f_{i,k}} \cdot \prod_{\mathsf{id}\in\mathcal{P}_{i,\Delta}} e\left(C_{\mathsf{id},i}, Y_{\mathsf{id}}\right)\cdot e\left(R_i, g_2\right)\right)^{g_i}$$
$$= \prod_{k=1}^{n} h_{l_k}^{f_k} \cdot \prod_{\mathsf{id}\in\mathcal{P}^*_\Delta} e\left(C^*_{\mathsf{id}}, Y^*_{\mathsf{id}}\right)\cdot e\left(R^*, g_2\right)$$

We also have $\prod_{\mathsf{id}\in\mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\prod_{i=1}^{N} C_{i,\mathsf{id}}^{g_i}, Y_{\mathsf{id}}\right)$

$$= e\left(\prod_{i=1}^{N} M_i^{g_i}, g_2\right) = e\left(M^*, g_2\right)$$

Thus all checks of $\mathsf{Ver}()$ pass and $\mathsf{Ver}(\mathcal{P}^*_\Delta, \mathsf{vk}, M^*, \sigma^*) = 1$ holds.

Finally it is an immediately corollary of Theorem 5.9 as well as Lemma 5.10 - 5.16, that no security reduction requires knowledge of messages m in $\mathbb{Z}_p$ but can be computed knowing group element $M = g_1^m \in \mathbb{G}_1$. Note that one of the modifications compared to MKLin (see Construction 5.3) authenticators are of the form $\sigma = (\Lambda, R)$ and not of the form $\sigma = (\Lambda, R, S)$. In Lemma 5.14 we dealt with a case, where $\mathcal{A}$ produced a forgery which contained a modification of the S component. As here no S component exists such a forgery is trivially impossible and we do not need to apply Lemma 5.14.

□

We recall that in Section 7.2 we showed how to obtain an FDC by combining a structure preserving homomorphic authenticator scheme with a conventional commitment scheme. We have now constructed a modification of our scheme MKLin (see Construction) 5.3) that is structure preserving.

We will provide the commitment scheme this can be combined with to obtain an FDC.

Construction 7.15.

$\mathsf{Setup}(1^\lambda)$: *On input a security parameter λ this algorithm chooses a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ as well as $H_0, \ldots, H_T \xleftarrow{\$} \mathbb{G}_1$ uniformly at random. It sets $\mathsf{CK} = (\mathsf{bgp}, H_0, \ldots, H_T)$. This is implicitly used in all the following algorithms.*

$\mathsf{Commit}(m, r)$: *On input a message $m \in \mathbb{Z}_p^T$, and randomness $r \in \mathbb{Z}_p$, it computes $C = H_0^r \cdot \prod_{j=1}^{T} H_j^{m[j]}$. It outputs the commitment C as well as the decommitment $d = r$.*

$\mathsf{Decommit}(m, d, C)$: *On input a message* $m \in \mathbb{Z}_p^T$*, decommitment* $d = r \in \mathbb{Z}_p$*, and a commitment* $C \in \mathbb{G}_1$ *it checks whether whether* $C = H_0^d \cdot \prod_{j=1}^T H_j^{m[j]}$*. If the equation holds it outputs '1', else it outputs '0'.*

Note that Commit and Decommit are identical to PublicCommit and PublicDecommit from Construction 4.13. Therefore the scheme also satisfies the same basic commitment properties - it is binding, hiding, and homomorphic.

Proposition 7.16. *The commitment scheme 7.15 is a binding commitment scheme under the DL assumption (see Def. 2.36).*

Proof. This is an immediate corollary of Theorem 4.15. □

Proposition 7.17. *The commitment scheme 7.15 is unconditionally hiding.*

Proof. This is an immediate corollary of Theorem 4.16.

□

Proposition 7.18. *The commitment scheme 7.15 is linearly homomorphic.*

Proof. If we have N commitments $C_1, \ldots, C_N$, N messages $m_1, \ldots, m_N$ and decommitments $d_1, \ldots d_N$ such that $C_i = H_0^{d_i} \cdot \prod_{j=1}^T H_j^{m_i[j]}$ holds for all $i \in [N]$, f is an arbitrary linear function given by its coefficient vector $(f_1, \ldots f_N)$ and we set $C = \prod_{i=1}^N C_i^{f_i}$, $m = \sum_{i=1}^N f_i m_i$, and $d = \sum_{i=1}^N f_i d_i$, then we have

$$\begin{aligned} C &= \prod_{i=1}^N C_i^{f_i} \\ &= \prod_{i=1}^N \left(H_0^{d_i} \cdot \prod_{j=1}^T H_j^{m_i[j]} \right)^{f_i} \\ &= H_0^{\sum_{i=1}^N f_i d_i} \cdot \prod_{j=1}^T H_j^{\sum_{i=1}^N f_i m_i[j]} \\ &= H_0^d \cdot \prod_{j=1}^T H_j^{m[j]}. \end{aligned}$$

Therefore $\mathsf{Decommit}(m, d, C) = 1$ holds. □

7.3.2 Multi-Key FDC Combined with Secret Sharing

We will now describe in detail the verifiable computing scheme we get by applying our transformation 7.5 to our homomorphic authenticator scheme from Construction 7.13, and combining it with Shamir secret sharing. This results in a scheme with both information-theoretic input and output privacy with respect to the verifier and information-theoretic input and output privacy with respect to the servers.

Construction 7.19.

$\mathsf{VKeyGen}(1^\lambda, \mathcal{P})$: *On input a security parameter* λ *and the description of a linear function* f *given as a labeled program* $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$*, it runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{sk}', \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, as well as* $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$*. It sets* $\mathsf{sk} = (\mathsf{sk}', \mathcal{P})$*,* $\mathsf{ek} = \mathcal{P}$*,* $\mathsf{vk} = (\mathsf{pk}, F)$ *and returns* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*.*

$\mathsf{ProbGen}(\mathsf{sk}, x)$*: On input the secret key* sk *and* $x = (m_1, \ldots, m_n, \Delta)$ *consisting of a tuple of* n *messages* $m_i \in \mathbb{Z}_p^T$ *for* $i \in [n]$ *and a dataset identifier* $\Delta \in \{0,1\}^*$*, it chooses* $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random. It applies Shamir secret sharing to each message entry* $m_i[j]$ *for all* $i \in [n], j \in [T]$ *as well as to all* r_i *for* $i \in [n]$*, i.e. it computes* $(s_1(m_i[j]), \ldots, s_N(m_i[j])) \leftarrow \mathsf{SShare}(m_i[j])$ *for all* $i \in [n], j \in [T]$*, as well as* $(s_1(r_i), \ldots, s_N(m_i)) \leftarrow \mathsf{SShare}(r_i)$ *for all* $i \in [n]$*. It runs* $A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau)$*. Additionally, it chooses* $k^* \in [N]$*. This will identify the distinguished shareholder that will perform operations on authenticators.* k^* *can be chosen according to a clients preferences. It outputs the shares* $s_k(m_i[j])$ *as well as* $s_k(r_i)$ *giving* $s_k(m_i[j])$ *to shareholder* k *and* $s_k(r_i)$ *to shareholder* k *for for* $i \in [n], j \in [T], k \in [N]$*, and additionally outputs* $A_1, \ldots, A_n$ *to shareholder* k^**. It sets* $\rho_x = 0$ *and* $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n], j\in[T], k\in[N]})$*. It outputs* (σ_x, ρ_x)*.*

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *On input an evaluation key* ek *and an encoded input* σ_x*, the algorithm parses* $\mathsf{ek} = (f, \tau_1, \ldots, \tau_n)$ *with* f *a linear function given by its coefficient vector* $(f_1, \ldots, f_n)$ *and* $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n], j\in[T], k\in[N]})$*. Each shareholder* k *computes* $s_k(m^*[j]) = \sum_{i=1}^n f_i \cdot s_k(m_i[j])$ *as well as* $s_k(r^*) = \sum_{i=1}^n f_i \cdot s_k(r_i)$*. They set* $s_k(m^*) = (s_k(m^*[1]), \ldots, s_k(m^*[T]))$*.*
Additionally k^* *runs* $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$*. It sets* $\sigma_y = (\Delta,\ A^*,\ \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$ *and outputs* σ_y*.*

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: *On input a verification key* vk*, a decoding value* ρ_x *and an encoded value* σ_y*, it parses* $\mathsf{vk} = (\mathsf{pk}, F)$*,* $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$*. It chooses a subset* $\mathcal{B} \subset [N]$ *with* $|\mathcal{B}| = t$*. It creates the reconstruction vector* $(w_1, \ldots, w_t)$ *derived from* $\mathcal{B}$ *(see Sec. 2.4 for details). It computes* $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$ *as well as* $r^* = \sum_{k\in\mathcal{B}} w_k s_k(r^*)$*. Then, it runs* $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$*. Finally, it runs* $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta)$*. If* $b = 0$ *it outputs* $\perp$*, else it outputs* m^**.*

Note that this is mostly analogous to Construction 4.27.

We now look at the basic properties of this construction. A first and obvious requirement is correctness, showing that any honest execution of the algorithm leads to verifiers accepting a correct result.

Proposition 7.20. *Construction 7.19 is a correct verifiable computing scheme ins*

the sense of Def. 2.30

Proof. Let f be an arbitrary linear function, $x = (m_1, \ldots, m_m, \Delta)$ be an arbitrary input. Let f be described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Let $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$, $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$, and $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$.

Let $y = \sum_{i=1}^{n} f_i m_i$. We parse $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in[N]})$ and $\sigma_y = (\Delta,\ A^*,\ \{s_k(m^*),\ s_k(r^*)\}_{k\in[N]})$. Following Verify we compute $C^* = \prod_{k\in\mathcal{B}} s_k(C^*)^{w_k}$. Then we have

$$\begin{aligned}
C^* &= \prod_{k\in\mathcal{B}} s_k(C^*)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(s_k(y), s_k(r^*))\right)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(s_k(\sum_{i=1}^{n} f_i m_i), s_k(\sum_{i=1}^{n} f_i r_i))\right)^{w_k} \\
&= \prod_{k\in\mathcal{B}} \left(\mathsf{PublicCommit}(\sum_{i=1}^{n} s_k(f_i m_i), \sum_{i=1}^{n} s_k(f_i r_i))\right)^{w_k}.
\end{aligned}$$

By the correctness of our FDC scheme (see Propositions 5.5 and 7.9) we have therefore $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) = 1$.

□

Next we consider the case of third party verifiers and show that this construction is even publicly verifiable.

Proposition 7.21. *Construction 7.19 is a publicly verifiable computing scheme.*

Proof. Note, that $\rho_x = 0$ by definition. Obviously this does not need to be kept secret. We have $\mathsf{vk} = (\mathsf{pk}, F)$, where pk is the public key of the FDC scheme and F is a function commitment. Both values are public. □

Now we formally show that this combination of our FDC with Shamir secret sharing does indeed lead to secure verifiable computing scheme. The security essentially follows from the unforgeability of the underlying FDC and its binding property.

Proposition 7.22. *Construction 7.19 is an adaptively secure verifiable computing scheme in the sense of Def. 2.33.*

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathbf{EXP}^{\mathsf{AdaptVerify}}_{\mathcal{A}}[\mathsf{VC}, f, \lambda]$ (see Def. 2.33), we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to either win the security experiment $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}$ (see Def. 4.9) or the security experiment $\mathbf{EXP}^{Bind}_{\mathcal{A},\mathrm{Com}}(\lambda)$ (see Def. 4.2).

Setup Simulator $\mathcal{S}$ runs $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$ and outputs pp. It chooses an arbitrary linear function f described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Let $x = (m_1, \ldots, m_n, \Delta)$ be an arbitrary input, let f be an arbitrary linear function and let $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$.

Key Generation: Simulator $\mathcal{S}$ runs $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. By construction, $\mathsf{ek} = 0$. Furthermore it runs $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. It sets $\mathsf{sk} = (\mathsf{sk}', \mathcal{P})$, $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, F)$ and returns $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$.

Queries: When $\mathcal{A}$ queries $x = (m_1, \ldots, m_n, \Delta)$ $\mathcal{S}$ does the following. It chooses $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random, and queries for $(\Delta, \tau_i, m_i, r_i)$ for $i \in [n]$, receiving A_i. It runs $\{s_k(m_i[j])\}_{k\in[N]} \leftarrow \mathsf{SShare}(m_i[j])$ for $i \in n, j \in [T]$ as well as $\{s_k(r_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(r_i)$ and outputs $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in[N]})$. Note that this is the identical response to an honest evaluation of $\mathsf{ProbGen}$.

Forgery: $\mathcal{A}$ returns σ_y^*. $\mathcal{S}$ parses $\sigma_y^* = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$. It chooses a subset $\mathcal{B}$ of size t and runs $m^* \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(m^*)\}_{k\in\mathcal{B}})$, as well as $r^* \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(r^*)\}_{k\in\mathcal{B}})$.
It computes $\hat{m} = \sum_{i=1}^n f_i m_i$ as well as $\hat{r} = \sum_{i=1}^n f_i r_i$. Then, it sets $\hat{C} \leftarrow \mathsf{PublicCommit}(\hat{m}, \hat{r})$ and $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$. It checks, whether $C^* = \hat{C}$. If it holds $(m^*, r^*), (\hat{m}, \hat{r})$ wins the binding experiment $\mathbf{EXP}^{Bind}_{\mathcal{A},\mathrm{Com}}(\lambda)$ (see Def. 4.2). If $C^* \neq \hat{C}$, it sets $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}, \Delta^*)$ and outputs $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$. If $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) \neq \perp$, then $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$ is a type 2 forgery as defined in Def. 4.8.

If we have $\mathbf{EXP}^{\mathsf{AdaptVerify}}_{\mathcal{A}}[\mathsf{VC}, f, \lambda] = 1$, then in particular we have $\sigma_y^* = (\Delta, A^*, \{s_k(C^*)\}_{k\in[N]})$ such that for $C^* = \prod_{k\in\mathcal{B}} s_k(C^*)^{w_k}$ we have $\mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta) = 1$, which implies a forgery in $\mathbf{EXP}^{UF-CMA}_{\mathcal{A},\mathrm{FDC}}(\lambda)$.

Therefore the claim follows from Theorem 5.9 Proposition 7.10, as well as Proposition 7.16. □

Furthermore we can show that our construction preserves efficient verification. After a one time function-dependent preprocessing verification can indeed be faster than a computation of the function itself.

Proposition 7.23. *Construction 7.19 is a verifiable computing scheme that achieves amortized efficiency in the sense of Def. 2.35.*

Proof. This is an immediate corollary of Propositions 5.7 and 7.12. □

Finally we show that our verifiable computing scheme achieves complete information theoretic privacy. Over the following four propositions we prove that it offers information-theoretic input privacy with respect to the servers, information-theoretic output privacy with respect to the servers, information-theoretic input

privacy with respect to the verifier and information-theoretic output privacy with respect to the verifier which have each been defined in Section 3.1.

Proposition 7.24. *Construction 7.19 achieves information-theoretic input privacy with respect to the servers in the sense of Def. 3.1 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:**
Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.
Let $(m_1, \ldots, m_n) \leftarrow \mathbb{Z}_p^T$ and $(m'_1, \ldots, m'_n) \leftarrow \mathbb{Z}_p^T$ be any two tuples of messages. Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$ for $i \in \{0,1\}$.
The adversary $\mathcal{A}$ chooses a subset $\mathcal{B} \subset [N]$ of size $|\mathcal{B}| = t-1$.
We assume $k^* \in \mathcal{B}$. If the adversary does not corrupt k^* the claim immediately follows from the hiding property of Shamir secret sharing [97].

Thus the adversary obtains and seeks to distinguish
$(x_0, x_1, \Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$
and
$(x_0, x_1, \Delta, \{A'_i, s_k(m'_i[j]), s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
By the hiding property of Shamir secret sharing
$(x_0, x_1, \Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$ is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A_i, R_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
Obviously this is perfectly indistinguishable from $(x_0, x_1, \Delta, \{A_i, R'_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$, as this is just another sampling of randomness. By the hiding property of the FDC (see Proposition 7.3.1), this is perfectly indistinguishable from $(x_0, x_1, \Delta, \{A'_i, R'_{ijk} \mid A'_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m'_i, r'_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$. Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A'_i, s_k(m'_i[j]), s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$. This completes the proof.
□

Proposition 7.25. *Construction 7.19 achieves information-theoretic output privacy with respect to the servers in the sense of Def. 3.2 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:** This setup is identical to Proposition 7.24.

Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y_0 = \sum_{i=1}^n f_i m_i$, $y_1 = \sum_{i=1}^n f_i m'_i$, $r = \sum_{i=1}^n f_i r_i$, $r' = \sum_{i=1}^n f_i r'_i$.
We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y_0), s_k(r)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y_1), s_k(r')\}_{k\in[N]})$.

Thus the adversary obtains and seeks to distinguish $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ and $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$.

By the hiding property of Shamir secret sharing $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R_k, S_k \mid R_k, S_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Obviously this is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R'_k, S'_k \mid R'_k, S'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$ as this is just another sampling of randomness. By the hiding property of the FDC (see Proposition 7.3.1), this is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A', R'_k, S'_k \mid R'_k, s'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$. This completes the proof.

□

Proposition 7.26. *Construction 7.19 achieves information-theoretic input privacy with respect to the verifier in the sense of Def. 3.3.*

Proof. Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

Let $m_1, \ldots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages. such that $\sum_{i=1}^n f_i m_i = \sum_{i=1}^n f_i m'_i$.
Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$. Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y = \sum_{i=1}^n f_i m_i$, $r = \sum_{i=1}^n f_i r_i$, and $r' = \sum_{i=1}^n f_i r'_i$. By assumption we have $y = \sum_{i=1}^n f_i m'_i$.

We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y), s_k(r)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y), s_k(r')\}_{k\in[N]})$. Thus the adversary obtains and seeks to distinguish $(x_0, x_1, \Delta, A, \{s_k(y), s_k(r)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(r')\}_{k\in\mathcal{B}})$.

Note that an adversary $\mathcal{A}$ that can distinguish $(x_0, x_1, \Delta, A, \{s_k(y), s_k(r)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(r')\}_{k\in\mathcal{B}})$ immediately implies an adversary $\mathcal{A}'$ that can distinguish (x_0, x_1, Δ, y, r) and $(x_0, x_1, \Delta, y, r')$. Since both r and r' are distributed uniformly at random as linear combinations of uniformly randomly chosen values, these are perfectly indistinguishable.

□

Proposition 7.27. *Construction 7.19 achieves information-theoretic output privacy with respect to the verifier in the sense of Def. 3.4.*

Proof. Let f be a linear function given by its coefficient vector $(f_1, \ldots, f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

We first show correctness in the sense of Def. 3.4. We provide the three additional algorithms.

$\mathsf{HideCompute}(\mathsf{ek}, \sigma_x)$: On input an evaluation key ek and an encoded input σ_x, the algorithm parses $\sigma_x = (\Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n], j\in[T], k\in[N]})$. Each

shareholder k computes $s_k(m^*[j]) = \sum_{i=1}^{n} f_i \cdot s_k(m_i[j])$, as well as $s_k(r^*) = \sum_{i=1}^{n} f_i \cdot s_k(r_i)$. They run $s_k(C^*) \leftarrow \mathsf{PublicCommit}(s_k(m^*), s_k(r^*))$. Additionally, k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$.

It sets $\tilde{\sigma_y} = (\Delta, A^*, \{s_k(C^*)\}_{k\in[N]})$ and outputs the encoded version $\tilde{\sigma_y}$.

$\mathsf{HideVerify}(\mathsf{vk}, \tilde{\sigma_y})$: On input a verification key $\mathsf{vk} = (\mathsf{pk}, F)$ and an encoded value $\tilde{\sigma_y}$, it parses $\tilde{\sigma_y} = (\Delta, A^*, \{s_k(C^*)\}_{k\in[N]})$. It chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| = t$. It creates the reconstruction vector $(w_1, \ldots, w_t)$ derived from $\mathcal{B}$ (see Sec. 2.4 for details). Then, it runs $C^* \leftarrow \mathsf{PublicCommit}(m^*, r^*)$. Finally, it runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta)$. If $b = 0$ it outputs $\perp$, else it sets $\hat{\sigma}_y = (\Delta, A^*, C^*)$ and outputs $\hat{\sigma}_y$.

$\mathsf{Decode}(\mathsf{vk}, \rho_x, \hat{\sigma}_y, \sigma_y)$: On input a verification key vk, a decoding value $\rho_x = 0$, and encoded values $\hat{\sigma}_y, \sigma_y$, it parses $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$ and $\hat{\sigma}_y = (\Delta, A^*, C^*)$. It computes $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$ as well as $r^* = \sum_{k\in\mathcal{B}} w_k s_k(r^*)$, and runs $b \leftarrow \mathsf{PublicDecommit}(C^*, m^*, r^*)$. If $b = 0$ it outputs $\perp$ else it returns m^*.

Now we show privacy in the sense of Def. 3.4.
Let $m_1, \ldots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages.
Setting $x_0 = (m_1, \ldots, m_n, \Delta)$ and $x_1 = (m'_1, \ldots, m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$.
Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0, 1\}$ and $y_0 = \sum_{i=1}^{n} f_i m_i$, $y_1 = \sum_{i=1}^{n} f_i m'_i$.
We parse $\sigma_{y_0} = (\Delta, A, \{s_k(C)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(C')\}_{k\in[N]})$.

Note that an adversary $\mathcal{A}$ that can distinguish $(y_0, y_1, \Delta, A, \{s_k(C)\}_{k\in[N]})$ and $(y_0, y_1, \Delta, A', \{s_k(C')\}_{k\in[N]})$ immediately implies an adversary $\mathcal{A}'$ that can distinguish (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$. However, (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$ are perfectly indistinguishable by the hiding property of the FDC (see Theorem 7.3.1).

□

7.4 Turning CHQS into an FDC

We now seek to apply the transformation described in Section 7.2 (Construction 7.5) to suitable homomorphic authenticator schemes. We recall that we presented CHQS (Construction 5.17) in Chapter 5 a homomorphic signature scheme supporting multivariate polynomials of degree 2. In the following, we present a slight variation of that scheme that, together with a novel homomorphic commitment scheme, will be used in our transformation. We recall that CHQS on its own provided information-theoretic input privacy with respect to the verifier. By applying our transformation and combining the resulting FDC with a suitable secret sharing scheme we obtain a verifiable computing scheme that provides information-theoretic input and output privacy with respect to the verifier and information-theoretic input and output privacy with respect to the servers.

7.4.1 A New Homomorphic Commitment Scheme

We will now describe a new commitment scheme which we will use for our transformation described in Sec. 7.2. Similar to CHQS, this commitment is *graded*. On a high level we provide two ways of committing to a message. One leading to so called level-1 commitments, the other to level-2 commitments. This leads to a homomorphic property beyond linear evaluation. Our homomorphic evaluation will in particular allow to evaluate two level-1 commitments to m_1 and m_2 respectively to produce a level-2 commitment to $m_1 m_2$.

Construction 7.28.

$\mathsf{Setup}(1^\lambda)$: *On input a security parameter λ this algorithm chooses a bilinear group* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$ *as well as* $H \xleftarrow{\$} \mathbb{G}_1$ *uniformly at random. It sets* $\mathsf{CK} = (\mathsf{bgp}, H)$. *This is implicitly used in all the following algorithms.*

$\mathsf{Commit}_1(m, y, r, r')$: *On input a message $m \in \mathbb{Z}_p$, and randomness $(y, r, r') \in \mathbb{Z}_p^3$, it computes $c = m + yr$, $C = g_1^m \cdot H^r \cdot g_1^{yr'}$, $Y = g_1^y$, as well as $R = g_2^r$, $R' = g_2^{r'}$. It outputs the level-1 commitment $Com = (c, C, Y)$ as well as the decommitment $d = (R, R')$.*

$\mathsf{Commit}_2(m, y, r, r')$: *On input a message $m \in \mathbb{Z}_p$, and randomness $(y, r, r') \in \mathbb{Z}_p^3$, it computes $C = g_1^m \cdot H^r \cdot g_1^{yr'}$, $Y = g_1^y$, as well as $R = g_2^r$, $R' = g_2^{r'}$. It outputs the level-2 commitment $Com = (C, Y)$ as well as the decommitment $d = (R, R')$.*

$\mathsf{Decommit}_1(m, d, Com)$: *On input a message $m \in \mathbb{Z}_p$, decommitment $d = (R, R') \in \mathbb{G}_2^2$, and a level -1 commitment $Com = (c, C, Y) \in \mathbb{Z}_p \times \mathbb{G}_1^2$ it checks whether $e(g_1^c, g_2) = e(g_1^m, g_2) \cdot e(Y, R)$ and $e(C, g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$. If both equations hold it outputs '1', else it outputs '0'.*

$\mathsf{Decommit}_2(m, d, Com)$: *On input a message $m \in \mathbb{Z}_p$, decommitment $d = (R, R') \in \mathbb{G}_2^2$, and a level -2 commitment $Com = (C, Y) \in \mathbb{G}_1^2$ it checks whether $e(C, g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$. If the equation holds it outputs '1', else it outputs '0'.*

The security of this scheme is based on the following variation of the DDH assumption.

Definition 7.29 (Augmented Decisional Diffie Hellmann Problem)**.** *Let $\mathcal{G}$ be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$. *We say the* augmented decisional Diffie Hellman *assumption (aug-DDH) holds in* bgp *if, for every PPT adversary $\mathcal{A}$, we have*

$$|\Pr\left[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_2^{\frac{1}{x}}, g_2^{xy}) \mid x, y \xleftarrow{\$} \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_2^{\frac{1}{x}}, g_2^z) \mid x, y, z \xleftarrow{\$} \mathbb{Z}_p\right]| = \mathsf{negl}(\lambda)$$

By the master theorem of [21] this assumption holds in the generic group model.

We will now prove the basic properties of this scheme. First we show the target binding property. This ensures that any commitment created by a simulator can only be opened to the original message by any computationally bounded adversary.

Proposition 7.30. *The commitment scheme 7.28 is a target binding commitment scheme in the sense of Def. 4.5 under the augmented DDH assumption (see Def. 7.29) and the* dba_1 *assumption (see Def. 2.40).*

Proof. To prove this statement we will describe two games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1' only with negligible probability. Following the notation of [37], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We use flag values bad_i, initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during game i.

Game 1: This game is just the security experiment 4.4 between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Game 2: This game is the same as game 1 except for the following change. $\mathcal{C}$ runs $(Com, d) \leftarrow \mathsf{Commit}_1(m, y, r, r')$ during the security experiment 4.4 and parses $d = (R, R')$. $\mathcal{C}$ parses d' returned by $\mathcal{A}$ as (S, S'). It checks whether $R = S$. If this holds it sets $\mathsf{bad}_2 = \mathsf{true}$.

We obviously have $|\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \Pr[\mathsf{bad}_2 = \mathsf{true}]$. We first show that any adversary that breaks the target binding property while $\mathsf{bad}_2 = \mathsf{true}$ implies a solver for the augmented DDH problem (see Def. 7.29). We show how a simulator $\mathcal{S}$ can use such an adversary. $\mathcal{S}$ takes as input $\mathsf{bgp}, g_1^\alpha, g_2^{\frac{1}{\alpha}}, g_1^\beta, g_2^\gamma$. It sets $\mathsf{bgp}' = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1^\alpha, g_2, e)$.

It sets $H = g_1$ and gives the commitment key (bgp', H) to $\mathcal{A}$.

It chooses $y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $Y = g_1^y$. It chooses an arbitrary message $m \in \mathcal{M} \subset \mathbb{Z}_p$, and randomness $r, r' \xleftarrow{\$} \mathbb{Z}_p$. It sets $c = m + yr$ as well as $C = (g_1^\alpha)^m \cdot (g_1^\alpha)^r \cdot g_1^{yr'}$. Furthermore it sets $R = \left(g_2^{\frac{1}{\alpha}}\right)^r$, $R' = g_2^{r'}$ and $d = (R, R')$. This is perfectly indistinguishable from an honest execution of the algorithm to $\mathcal{A}$.

Note that we have

$$e\left(C, g_2\right) = e\left(\left(g_1^\alpha\right)^m, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right)$$

It gives (C, m, d) to the adversary $\mathcal{A}$.

The adversary $\mathcal{A}$ returns (m', d') with $m \neq m'$. $\mathcal{S}$ parses $d' = (S, S') \in \mathbb{G}_2^2$.

If $\mathcal{A}$ wins the security game we have

$$e\left(C, g_2\right) = e\left(\left(g_1^\alpha\right)^{m'}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right).$$

Dividing the equations and using the fact that $R = S$ we obtain

$$1 = e\left((g_1^{\alpha})^{m-m'}, g_2\right) \cdot e\left(Y, \frac{R'}{S'}\right)$$

or equivalently

$$1 = e\left((g_1^{\alpha})^{m'-m}, g_2\right) \cdot e\left(g_1, \frac{R'^{y}}{S'}\right).$$

Since we know that $m \neq m'$ we have $g_2^{\alpha} = \left(\frac{R'}{S'}\right)^{\frac{y}{m'-m}}$.

We now have $\gamma = \alpha\beta$ if and only if

$$e(g_1, g_2^{\gamma}) = e\left(g_1^{\beta}, \left(\frac{R'}{S'}\right)^{\frac{y}{m'-m}}\right)$$

holds.

Now it remains to show that any adversary winning Game 2 can be used to solve the DBP_1 problem (see Def. 2.40).

We show how a simulator $\mathcal{S}$ can use such an adversary.

$\mathcal{S}$ takes as input $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_1^x, g_2, e)$. It seeks to compute $G, G_x \in \mathbb{G}_2$ such that $1 = e(g_1, G) \cdot e(g_1^x, G_x)$.

It sets $H = g_1^x$ as wellas $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ and gives the commitment key (bgp, H) to $\mathcal{A}$.

It chooses $y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $Y = g_1^y$. It chooses an arbitrary message $m \in \mathcal{M} \subset \mathbb{Z}_p$, and randomness $r, r' \xleftarrow{\$} \mathbb{Z}_p$. It sets $c = m + yr$ as well as $C = g_1^m \cdot g_1^r \cdot H^r \cdot g_1^{yr'}$. Furthermore it sets $R = g_2^r$, $R' = g_2^{r'}$ and $d = (R, R')$. The output $((c, C, Y), (R, R'))$ is perfectly indistinguishable from an honest execution of Commit_1 (see Construction 7.28) to $\mathcal{A}$. Note that we have

$$e(C, g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$$

It gives (C, m, d) to the adversary $\mathcal{A}$.

The adversary $\mathcal{A}$ returns (m', d') with $m \neq m'$. $\mathcal{S}$ parses $d' = (S, S') \in \mathbb{G}_2^2$.

If $\mathcal{A}$ wins the security game we have

$$e(C, g_2) = e\left(g_1^{m'}, g_2\right) \cdot e(H, S) \cdot e(Y, S').$$

Dividing the equations yields

$$1 = e\left(g_1^{m-m'}, g_2\right) \cdot e\left(H, \frac{R}{S}\right) \cdot e\left(Y, \frac{R'}{S'}\right)$$

or equivalently

$$1 = e\left(g_1, g_2^{m-m'} \cdot \frac{R'^{y}}{S'}\right) \cdot e\left(g_1^x, \frac{R}{S}\right)$$

We know that $R \neq S$. Therefore $G = g_2^{m-m'} \cdot \frac{R'^{y}}{S'}$ and $G_x = \frac{R}{S}$ is a valid solution for the DBP_1 problem.

□

Next we investigate the hiding property of our scheme. Note, that this will be the basis of the hiding property of the FDC we construct and thus ultimately allow us to achieve information-theoretic input privacy with respect to the verifier, information-theoretic output privacy with respect to the verifier, and information-theoretic input and output privacy with respect to the servers.

Proposition 7.31. *The commitment scheme 7.28 is unconditionally hiding.*

Proof. If $r \xleftarrow{\$} \mathbb{Z}_p$ is chosen uniformly at random then $\{m + yr \mid r \xleftarrow{\$} \mathbb{Z}_p\}$ is uniformly distributed over $\mathbb{Z}_p$ for every $m \in \mathbb{Z}_p$ and every $y \in \mathbb{Z}_p^*$. If $r' \xleftarrow{\$} \mathbb{Z}_p$ is chosen uniformly at random then $\{g_1^{r'} \mid r' \xleftarrow{\$} \mathbb{Z}_p\}$ is uniformly distributed over $\mathbb{G}_1$. Therefore $\{g_1^m \cdot H^r \cdot g_1^{yr'} | r' \xleftarrow{\$} \mathbb{Z}_p\}$ is uniformly distributed over $\mathbb{G}_1$ for all $m \in \mathbb{Z}_p$, $y \in \mathbb{Z}_p^*$, $r \in \mathbb{Z}_p$.

So in particular $\{m+r, g_1^m \cdot H^r \cdot g_1^{yr'} | r, r' \xleftarrow{\$} \mathbb{Z}_p\}$ and $\{m'+ys, g_1^{m'} \cdot H^s \cdot g_1^{s'} | s, s' \xleftarrow{\$} \mathbb{Z}_p\}$ are both uniformly distributed over $\mathbb{Z}_p \times \mathbb{G}_1$ for every possible choice of $m, m' \in \mathbb{Z}_p$. They are therefore identically distributed and perfectly indistinguishable.

□

Finally, we how the homomorphic property of our scheme. We recall that in our scheme CHQS (Construction 5.17) we described homomorphic evaluation for *graded* authenticators by using six subalgorithms (Add_1, Mult, Add_2, cMult_1, cMult_2, Shift). In this chapter we will follow this structure.

In Proposition 7.32 we describe these algorithms for the homomorphic evaluation of (ordinary) commitments. In Construction 7.33 we describe these algorithms for FDCs. Finally, in Construction 7.35 we describe these algorithm for a verifiable MPC scheme. For easy of notation we will denote these algorithms by the same name, where the exact procedure depends on the objects (commitmetns, FDCs, encoded inputs) given to the algorithm.

Proposition 7.32. *The commitment scheme 7.28 allows for the homomorphic evaluation of quadratic multivariate polynomials.*

Proof. We now show the correctness of the following six subalgorithms.

Add_1: On input two level-1 commitments $Com_i = (c_i, C_i, Y)$ for $i = 1, 2$, it sets $c = c_1 + c_2$, $C = C_1 \cdot C_2$, and outputs the level-1 commitment $Com = (c, C, Y)$.

cMult_1: On input a level-1 commitment $Com' = (c', C', Y)$ and a scalar $a \in \mathbb{Z}_p$ it sets $c = ac'$, $C = (C')^a$ and outputs the level-1 commitment $Com = (c, C, Y)$.

Mult: On input two level-1 commitments $Com_i = (c_i, C_i, Y)$ for $i = 1, 2$, it sets $C = C_1^{c_2}$ and outputs the level-2 commitment $Com = (C, Y)$.

Add_2: On input two level-21 commitments $Com_i = (C_i, Y)$ for $i = 1, 2$, it sets $C = C_1 \cdot C_2$, and outputs the level-2 commitment $Com = (C, Y)$.

cMult_2: On input a level-2 commitment $Com' = (2C', Y)$ and a scalar $a \in \mathbb{Z}_p$ it sets $C = (C')^a$ and outputs the level-2 commitment $Com = (C, Y)$.

Shift: On input a level-1 commitment $Com' = (c', C', Y)$, it sets $C = C'$ and outputs the level-2 commitment $Com = (C, Y)$.

We will now deal with the correctness of each subalgorithm.

Add_1: If we have two level-1 commitments $Com_i = (c_i, C_i, Y)$ for $i = 1, 2$, messages, m_1, m_2 decommitments d_1, d_2 with $d_1 = (R, R')$ and $d_2 = (S, S')$ such that

$$\begin{aligned} e\left(g_1^{c_1}, g_2\right) &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(Y, R\right) \\ e\left(C_1, g_2\right) &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right) \\ e\left(g_1^{c_2}, g_2\right) &= e\left(g_1^{m_2}, g_2\right) \cdot e\left(Y, S\right) \\ e\left(C_2, g_2\right) &= e\left(g_1^{m_2}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right) \end{aligned}$$

all hold, then we have

$$\begin{aligned} e\left(g_1^{c}, g_2\right) &= e\left(g_1^{c_1+c_2}, g_2\right) \\ &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(Y, R\right) \cdot e\left(g_1^{m_2}, g_2\right) \cdot e\left(Y, S\right) \\ &= e\left(g_1^{m_1+m_2}, g_2\right) \cdot e\left(Y, RS\right) \end{aligned}$$

as well as

$$\begin{aligned} &e\left(C, g_2\right) = e\left(C_1 \cdot C_2, g_2\right) = e\left(C_1, g_2\right) \cdot e\left(C_2, g_2\right) \\ &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right) \cdot e\left(g_1^{m_2}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right) \\ &= g_1^{m_1+m_2} \cdot e\left(H, RS\right) \cdot e\left(Y, R'S'\right). \end{aligned}$$

If we set $d = (RS, R'S')$, we therefore have for $Com = (c_1 + c_2, C_1 \cdot C_2, Y)$, $\mathsf{Decommit}_1(m_1 + m_2, d, Com) = 1$.

cMult_1: If we have a level-1 commitment $Com' = (c', C', Y)$ and $a \in \mathbb{Z}_p$, a message m and decommitment $d' = (R, R')$ such that

$$\begin{aligned} e\left(g_1^{c'}, g_2\right) &= e\left(g_1 m, g_2\right) \cdot e\left(Y, R\right) \\ e\left(C', g_2\right) &= e\left(g_1^{m}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, S\right) \end{aligned}$$

all hold, then we have

$$\begin{aligned} e\left(g_1^c, g_2\right) &= e\left(g_1^{ac'}, g_2\right) \\ &= e\left(g_1^{am}, g_2\right) \cdot e\left(Y, R^a\right) \end{aligned}$$

as well as

$$\begin{aligned} e\left(C, g_2\right) &= e\left(C'^a, g_2\right) \\ &= e\left(g_1^{am}, g_2\right) \cdot e\left(H, R^a\right) \cdot e\left(Y, R'^a\right). \end{aligned}$$

If we set $d = (R^a, R'^a)$, we therefore have for $Com = (ac', C'^a, Y)$, $\mathsf{Decommit}_1(am, d, Com) = 1$.

Mult: If we have two level-1 commitments $Com_i = (c_i, C_i, Y)$ for $i = 1, 2$, messages m_1, m_2, decommitments d_1, d_2 with $d_1 = (R, R')$ and $d_2 = (S, S')$ such that

$$\begin{aligned} e\left(g_1^{c_1}, g_2\right) &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(Y, R\right) \\ e\left(C_1, g_2\right) &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right) \\ e\left(g_1^{c_2}, g_2\right) &= e\left(g_1^{m_2}, g_2\right) \cdot e\left(Y, S\right) \\ e\left(C_2, g_2\right) &= e\left(g_1^{m_2}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right) \end{aligned}$$

respectively hold, then we have

$$\begin{aligned} e\left(C, g_2\right) &= e\left(C_1^{c_2}, g_2\right) \\ &= e\left(g_1^{m_1 c_2}, g_2\right) \cdot e\left(H, R^{c_2}\right) \cdot e\left(Y, R'^{c_2}\right) \\ &= e\left(g_1^{m_1 m_2}, g_2\right) \cdot e\left(Y, S\right) \cdot e\left(H, R^{c_2}\right) \cdot e\left(Y, R'^{c_2}\right) \\ &= e\left(g_1^{m_1 m_2}, g_2\right) \cdot e\left(H, R^{c_2}\right) \cdot e\left(Y, R'^{c_2} \cdot S\right). \end{aligned}$$

If we set $d = (R^{c_2}, R'^{c_2} \cdot S)$ we therefore have for $Com = (C_1^{c_2}, Y)$, $\mathsf{Decommit}_2(m_1 \cdot m_2, d, Com) = 1$.

Add_2: If we have two level-2 commitments $Com_i = (C_i, Y)$ for $i = 1, 2$, messages, m_1, m_2 decommitments d_1, d_2 with $d_1 = (R, R')$ and $d_2 = (S, S')$ such that $e\left(C_1, g_2\right) = e\left(g_1^{m_1}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right)$, and $e\left(C_2, g_2\right) = e\left(g_1^{m_2}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right)$ respectively hold, then we have

$$\begin{aligned} e\left(C, g_2\right) &= e\left(C_1 \cdot C_2, g_2\right) = e\left(C_1, g_2\right) \cdot e\left(C_2, g_2\right) \\ &= e\left(g_1^{m_1}, g_2\right) \cdot e\left(H, R\right) \cdot e\left(Y, R'\right) \cdot e\left(g_1^{m_2}, g_2\right) \cdot e\left(H, S\right) \cdot e\left(Y, S'\right) \\ &= g_1^{m_1 + m_2} \cdot e\left(H, RS\right) \cdot e\left(Y, R'S'\right). \end{aligned}$$

If we set $d = (RS, R'S')$, we therefore have for $Com = (C_1 \cdot C_2, Y)$, $\mathsf{Decommit}_2(m_1 + m_2, d, Com) = 1$.

$\mathsf{cMult_2}$**:** If we have a level-1 commitment $Com' = (c', C', Y')$ and $a \in \mathbb{Z}_p$, a message m and decommitment $d' = (R, R')$ such that $e(C', g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$ holds, then we have

$$\begin{aligned} e(C, g_2) &= e\left(C'^a, g_2\right) \\ &= e(g_1^{am}, g_2) \cdot e(H, R^a) \cdot e\left(Y, R'^a\right). \end{aligned}$$

If we set $d = (R^a, R'^a)$, we therefore have for $Com = (C'^a, Y)$, $\mathsf{Decommit_2}(am, d, Com) = 1$.

Shift**:** If we have a level-1 commitment $Com' = (c', C', Y')$, a message m and decommitment $d' = (R, R')$ such that $e\left(g_1^{c'}, g_2\right) = e(g_1^m, g_2) \cdot e(Y, R)$, and $e(C', g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$ hold, then we have $e(C, g_2) = e(C', g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$. If we set $d = (R, R')$, we therefore have for $Com = (C', Y)$, $\mathsf{Decommit_2}(m, d, Com) = 1$.

□

7.4.2 A Structure Preserving Variant of CHQS

We will now discuss how CHQS (see Construction 5.17) can be interpreted as structure preserving over the structure $(\mathbb{Z}_p \times \mathbb{G}_1^2)$ with the homomorphic structure detailed in Prop. 7.32. Note that we are extending the notion of the structure preserving property analogously to Libert et al. [78]. More precisely, we will provide a homomorphic authenticator very similar to Construction 5.17. The main difference is the domain of the inputs to Auth. In CHQS messages were elements of $\mathbb{Z}_p$. In this variant, messages are taken from $\mathbb{Z}_p \times \mathbb{G}_1$. In the original CHQS the message $m \in \mathbb{Z}_p$ is a part of the authenticator and the component Λ is computed using g_1^m. Note that we have $\mathsf{Commit_1}(m, y, 0, 0) = (m, g_1^m, g_1^y)$. So the original CHQS can be seen as taking non-randomized (and therefore not hiding) commitments as inputs. In the following we will detail the more general case of randomized commitments, $\mathsf{Commit_1}(m, y, r, r') = (c, C, g_1^y)$, where $r, r' \xleftarrow{\$} \mathbb{Z}_p$ are taken uniformly at random.

Construction 7.33 (Sp-CHQS)**.**

$\mathsf{Setup}(1^\lambda)$*: On input a security parameter λ the algorithm runs $\mathcal{G}(1^\lambda)$ to obtain a bilinear group* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$*. It chooses $n \in \mathbb{N}$. Additionally, it fixes a regular signature scheme* $\mathsf{Sig} = (\mathsf{KeyGen_{Sig}}, \mathsf{Sign_{Sig}}, \mathsf{Ver_{Sig}})$ *with signature space Σ contained in the bilinear group. More precisely we have $\Sigma \subset \mathbb{G}_1^{l_1} \times \mathbb{G}_2^{l_2} \times \mathbb{G}_T^{l_T}$ for some $(l_1, l_2, l_T) \in \mathbb{N}_0^3$. Then, it fixes a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It chooses $H \xleftarrow{\$} \mathbb{G}_1$ uniformly at random. It outputs the public parameters* $\mathsf{pp} = (\lambda, n, \mathsf{bgp}, H, \mathsf{Sig}, \Phi)$.

$\mathsf{KeyGen}(\mathsf{pp})$ **:** *On input public parameters* pp *it chooses* $x, y \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random. It sets* $h_2 = g_2^x$*. It samples* $t_{\tau_i}, k_{\tau_i} \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random for all* $i \in [n]$ *and sets* $F_{\tau_i} = g_2^{t_{\tau_i}}$*, as well as* $f_{\tau_i} = g_t^{yt_{\tau_i}}$*,* $f_{\tau_i,\tau_j} = g_t^{t_{\tau_i}k_{\tau_j}}$*, for all* $i, j \in [n]$*. Additionally the algorithm chooses a random seed* $K \xleftarrow{\$} \mathcal{K}$ *for the pseudorandom function* Φ*. It computes keys for the regular signature scheme* $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*. It sets* $\mathsf{sk} = (\mathsf{sk}_{\mathsf{Sig}}, K, x, y, \{t_{\tau_i}\}_{i=1}^n, \{k_{\tau_i}\}_{i=1}^n)$*,* $\mathsf{ek} = 0$ *and* $\mathsf{vk} = (\mathsf{pk}_{\mathsf{Sig}}, h_2, \{F_{\tau_i}, f_{\tau_i}\}_{i=1}^n, \{f_{\tau_i,\tau_j}\}_{i,j=1}^n)$*.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m, M)$**:** *On input a secret key* sk*, a dataset identifier* Δ*, an input identifier* $\tau \in \mathcal{T}$*, and a message* $(m, M) \in \mathbb{Z}_p \times \mathbb{G}_1$*, the algorithm generates the parameters for the dataset identified by* Δ*, by running* $z \leftarrow \Phi_K(\Delta)$ *and computing* $Z = g_2^{\frac{1}{z}}$*.* Z *is bound to the dataset identifier* Δ *by using the regular signature scheme, i.e. it sets* $\sigma_\Delta \leftarrow \mathsf{Sign}_{\mathsf{Sig}}(\Delta||Z)$*. It chooses* $r, s \in \mathbb{Z}_p$ *uniformly at random. Then it computes* $\Lambda \leftarrow g_1^{z(xm+(y+s)t_\tau+r)}$*,* $\Lambda \leftarrow M^{xz} \cdot g_1^{z((y+s)t_\tau+r)}$*,* $R \leftarrow g_1^r$*,* $S_\tau \leftarrow g_1^s$*, as well as* $T_\tau \leftarrow g_1^{ym-k_\tau}$*. It sets* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ *and then returns the signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$*.*

$\mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$**:** *Inputs are a public evaluation key* ek*, an arithmetic circuit* f *of degree at most 2, and signatures* $\sigma_1, \ldots, \sigma_n$*, where (w.l.o.g.)* $\sigma_i = (m_i, \sigma_{\Delta,i}, Z_i, \Lambda_i, R_i, \mathcal{T}_i)$*. The algorithm checks if the signatures share the same public values, i.e. if* $\sigma_{\Delta,1} = \sigma_{\Delta,i}$ *and* $Z_1 = Z_i$ *for all* $i = 2, \ldots, n$*, and the signature for each set of public values is correct and matches the dataset identifier* Δ*, i.e.* $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta_i||Z_i) = 1$ *for any* $i \in [n]$*. If this is not the case, the algorithm rejects the signature. Otherwise, it proceeds as follows. We describe this algorithm in terms of six different procedures* $(\mathsf{Add}_1, \mathsf{Mult}, \mathsf{Add}_2, \mathsf{cMult}_1, \mathsf{cMult}_2, \mathsf{Shift})$ *allowing to evaluate the circuit gate by gate.*

- Add_1**:** *On input two level-1 signatures* $\sigma_i = (m_i, \sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{T}_i)$ *for* $i = 1, 2$ *it computes as follows:* $m = m_1 + m_2$*,* $\Lambda = \Lambda_1 \cdot \Lambda_2$*,* $R = R_1 \cdot R_2$*, and* $S_\tau = S_{\tau,1} \cdot S_{\tau,2}$ *as well as* $T_\tau = T_{\tau,1} \cdot T_{\tau,2}$ *for all* $(\tau, \cdot) \in \mathcal{T}_1 \cap \mathcal{T}_2$*,* $S_\tau = S_{\tau,i}$ *as well as* $T_\tau = T_{\tau,i}$ *for all* τ *such that* $(\tau, \cdot) \in \mathcal{T}_1 \Delta \mathcal{T}_2$*, and* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}$ *for all* $(\tau, \cdot) \in \mathcal{T}_1 \cup \mathcal{T}_2$*. It outputs a level-1 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$*.*
- Mult**:** *On input two level-1 signatures* $\sigma_i = (m_i, \sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{T}_i)$ *for* $i = 1, 2$ *and the public verification key* vk*, it computes as follows:* $\Lambda = \Lambda_1^{m_2}$*,* $R = R_1^{m_2}$*,* $S'_{\tau_1} = S_{\tau_1}^{m_2} \cdot \prod_{\tau_2 \in \mathcal{T}_2} T_{\tau_2}$*, for all* $\tau_1 \in \mathcal{T}_1$*, and* $\mathcal{L} = \{(\tau, S'_\tau)\}$ *for all* $\tau \in \mathcal{T}_1$*. It outputs a level-2 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*.*
- Add_2**:** *On input two level-2 signatures* $\sigma_i = (\sigma_\Delta, Z, \Lambda_i, R_i, \mathcal{L}_i)$ *for* $i = 1, 2$*, it computes as follows:* $\Lambda = \Lambda_1 \cdot \Lambda_2$*,* $R = R_1 \cdot R_2$*,* $S_\tau = S_{\tau,1} \cdot S_{\tau,2}$ *for all* $(\tau, \cdot) \in \mathcal{L}_1 \cap \mathcal{L}_2$*,* $S_\tau = S_{\tau,i}$ *for all* τ *such that* $(\tau, \cdot) \in \mathcal{L}_1 \Delta \mathcal{L}_2$*, and* $\mathcal{L} = \{(\tau, S_\tau)\}$ *for all* $(\tau, \cdot) \in \mathcal{L}_1 \cup \mathcal{L}_2$*. It outputs a level-2 signature* $\sigma = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*.*

cMult_1*: On input a level-1 signature* $\sigma' = (m', \sigma_\Delta, Z, \Lambda', R', \mathcal{T}')$ *and a constant* $c \in \mathbb{Z}_p$*, it computes as follows:* $m = cm'$*,* $\Lambda = \Lambda'^c$*,* $R = R'^c$*,* $S_\tau = S'^c_\tau$*,* $T_\tau = T'^c_\tau$ *for all* $(\tau, S'_\tau, T'_\tau) \in \mathcal{T}'$*, and* $\mathcal{T} = \{(\tau, S_\tau, T_\tau)\}_{\tau \in \mathcal{T}}$*. It outputs a level-1 signature* $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$*.*

cMult_2*: On input a level-2 signature* $\sigma = (\sigma_\Delta, Z, \Lambda', R', \mathcal{L}')$ *and a constant* $c \in \mathbb{Z}_p$*, it computes as follows:* $\Lambda = \Lambda'^c$*,* $R = R'^c$*,* $S_\tau = S'^c_\tau$ *for all* $(\tau, S'_\tau) \in \mathcal{L}'$*, and* $\mathcal{L} = \{(\tau, S_\tau)\}$ *for all* $(\tau, S'_\tau) \in \mathcal{L}'$*. It outputs a level-2 signature* $\sigma = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*.*

Shift*: On input a level-1 signature* $\sigma' = (m', \sigma_\Delta, Z, \Lambda', R', \mathcal{T}')$*, it sets* $\Lambda = \Lambda'$*,* $R = R'$*, and* $\mathcal{L} = \{(\tau, S_\tau)\}_{\tau \in \mathcal{T}'}$*. It outputs a level-2 signature* $\sigma = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*.* Shift *simply describes how to derive a level-2 signature from a level-1 signature.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, M, \sigma)$*: On input a public evaluation key* vk*, level-2 message* M*, a (level-1 or -2) signature* σ*, a multi-labeled program* $\mathcal{P}_\Delta$ *containing an arithmetic circuit* f *of degree at most 2, the algorithm parses (without loss of generality)* $\sigma = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$*. IF* Ver *receives as input a level-1 message* $(m, M) \in \mathbb{Z}_p \times \mathbb{G}_1$ *it applies* Shift *on both* (m, M) *and* σ *before verification.*
It then checks whether the following conditions hold:

1. $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta || Z) = 1$
   as well as

$$e(\Lambda, Z) = e(R, g_2) \cdot e(M, h_2) \cdot \prod_{i,j=1}^{n} f_{i,j}^{c_{i,j}} \cdot \prod_{j=1}^{n} f_j^{c_j} \cdot \prod_{(\tau,\cdot) \in \mathcal{L}} e(S_\tau, F_\tau)$$

   where $c_{i,j}$ *and* c_j *are the coefficients in* $\mathcal{P}_\Delta$*.*

If both conditions hold respectively, it returns '1'. Otherwise, it returns '0'.

Theorem 7.34. *SP-CHQS (see 7.33) is structure preserving over the structure* $(\mathsf{bgp} \times \mathbb{Z}_p)$ *with the homomorphic structure detailed in Proposition 7.32 if* Sig *produces signatures in* bgp*.*

Proof. First we see that both inputs and signatures lie in $\mathsf{bgp} \times \mathbb{Z}_p$ if the signature space Σ of Sig lies in bgp, i.e. there exists a tuple $(l_1, l_2, l_T) \in \mathbb{N}_0^3$, such that $\Sigma \subset \mathbb{G}_1^{l_1} \times \mathbb{G}_2^{l_2} \times \mathbb{G}_T^{l_T}$.

Second we show that authentication correctness in the sense of Def. 2.5 holds for SP-CHQS as well.

To that end we let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, $\tau \in \mathcal{T}$ an arbitrary label, $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier, and $(m, M) \in \mathbb{F}_p \times \mathbb{G}_1$ an arbitrary message. Furthermore let $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \tau, m)$. We parse $\sigma = (m, \sigma_\Delta, Z, \Lambda, R, \mathcal{T})$. As both (m, M) and σ are level-1 , we apply Shift to obtain the message M and the signature $\sigma' = (\sigma_\Delta, Z, \Lambda, R, \mathcal{L})$. with $\mathcal{L} = \{(\tau, S_\tau)\}_{\tau \in \mathcal{T}'}$.

If Sig is a correct signature scheme we have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. Then we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(M^{xz} \cdot g_1^{(y+s)t_\tau + r)}, g_2^{\frac{1}{z}}\right) \\ &= e\left(M^x \cdot g_1^{(y+s)t_\tau + r}, g_2\right) \\ &= e(M, g_2^x) \cdot g_t^{(y+s)t_\tau + r} \\ &= e(M, h_2) \cdot f_\tau \cdot e(R, g_2) \cdot e(S, F_\tau) \end{aligned}$$

Therefore all checks of Ver pass.

Third we show that evaluation correctness in the sense of Def. 2.6 holds for SP-CHQS as well. To that end we let λ be an arbitrary security parameter, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be arbitrary public parameters, $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ an arbitrary key triple, and $\Delta \in \{0,1\}^*$ an arbitrary dataset identifier.

We show the evaluation correctness of the six procedures (Add_1, Mult, Add_2, cMult_1, cMult_2, Shift). So we take any two program/message/authenticator triples $\{(\mathcal{P}_i, M_i, \sigma_i)\}_{i\in[2]}$, such that $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, M_i, \sigma_i) = 1$.

Add_1: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, (m_i, M_i), \sigma_i) = 1$ for $i = 1, 2$, we know that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $M_1 \cdot M_2 = g(M_1, M_2)$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e(\Lambda_1 \cdot \Lambda_2, Z_1) \\ &= e(\Lambda_1, Z_1) \cdot e(\Lambda_2, Z_2) \\ &= e(R_1, g_2) \cdot e(M_1, h_2) \cdot f_1 \cdot e(S_1, F_1) \cdot e(R_2, g_2) \cdot e(M_1, h_2) \cdot f_2 \cdot e(S_2, F_2) \\ &= e(R_1 \cdot R_2, g_2) \cdot e(M_1 \cdot M_2, h_2) \cdot f_1 \cdot f_2 \cdot e(S_1, F_1) \cdot e(S_2, F_2) \\ &= e(R, g_2) \cdot e(M_1) \cdot f_1 \cdot f_2 \cdot e(S_1, F_1) \cdot e(S_2, F_2) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

Mult: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, (m_i, M_i), \sigma_i) = 1$ for $i = 1, 2$, we know that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. For ease of Notation we consider the case where $\mathcal{T}_i$ each contains only a single entry. We also have $M_1^{m_2} = g(M_1, m_2)$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e(\Lambda_1^{m_2}, Z_1) = e(\Lambda_1, Z_1)^{m_2} \\ &= e(R_1^{m_2}, g_2) \cdot e(M_1^{m_2}, h_2) \cdot f_1^{m_2} \cdot e(S_1^{m_2}, F_1) \\ &= e(R_1^{m_2}, g_2) \cdot e(M_1^{m_2}, h_2) \cdot f_1^{m_2} \cdot e(S_1^{m_2}, F_1) \cdot e(T_2, F_1) \cdot f_{1,2} \cdot f_1^{-m_2} \\ &= e(R, g_2) \cdot e(M, h_2) \cdot f_{1,2} \cdot e(S_1^{m_2} \cdot T_2, F_1) \\ &= e(R, g_2) \cdot e(M, h_2) \cdot f_{1,2} \cdot e(S, F_1) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

Add_2: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{i,\Delta}, M_i, \sigma_i) = 1$ for $i = 1, 2$, we know in particular that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_{\Delta,i}, \Delta||Z_i) = 1$ for $i = 1, 2$. So with $Z = Z_1$, $\sigma_\Delta = \sigma_{\Delta,1}$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $M_1 \cdot M_2 = g(m_1, m_2)$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e(\Lambda_1 \cdot \Lambda_2, Z_1) = e(\Lambda_1, Z_1) \cdot e(\Lambda_2, Z_2) \\ &= e(R_1, g_2) \cdot e(M_1, h_2) \cdot f_1 \cdot \prod_{(\tau,\cdot)\in\mathcal{L}_1} e(S_{\tau,1}, F_\tau) \\ &\quad \cdot e(R_2, g_2) \cdot e(M_2, h_2) \cdot f_2 \cdot \prod_{(\tau,\cdot)\in\mathcal{L}_2} e(S_{\tau,2}, F_\tau) \\ &= e(R, g_2)\, e(M_1 \cdot M_2, h_2) \cdot f_1 \cdot f_2 \prod_{(\tau,\cdot)\in\mathcal{L}} e(S_\tau, F_\tau) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

cMult_1: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}'_\Delta, (m', M'), \sigma') = 1$, we know in particular that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$. So with $Z = Z'$, $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $M = M'^c = g(M')$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(\Lambda'^c, Z'\right) = e(\Lambda', Z')^c \\ &= e\left(R'^c, g_2\right) \cdot e\left(M'^c, h_2\right) \cdot f'^c \cdot e\left(S'^c, F\right) \\ &= e(R, g_2) \cdot e(M, h_2) \cdot f \cdot e(S, F) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

cMult_2: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}'_\Delta, M', \sigma') = 1$, we know in particular that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$. So with $Z = Z'$ and $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $M = M'^c = g(M')$. Furthermore we have

$$\begin{aligned} e(\Lambda, Z) &= e\left(\Lambda'^c, Z'\right) = e(\Lambda', Z')^c \\ &= e\left(R'^c, g_2\right) \cdot e\left(M'^c, h_2\right) \cdot f'^c \cdot e\left(S'^c, F\right) \\ &= e(R, g_2) \cdot e(M, h_2) \cdot f \cdot e(S, F) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

Shift: Since we have $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}'_\Delta, (m', M'), \sigma') = 1$, we know in particular that $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma'_\Delta, \Delta||Z') = 1$. So with $Z = Z'$ and $\sigma_\Delta = \sigma'_\Delta$ we also have $\mathsf{Ver}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, \sigma_\Delta, \Delta||Z) = 1$. We also have $M = M' = g(m', M')$. Furthermore

we have

$$\begin{aligned} e(\Lambda, Z) &= e(\Lambda', Z') \\ &= e(R', g_2) \cdot e(M', h_2) \cdot f' \cdot e(S', F) \\ &= e(R, g_2) \cdot e(M, h_2) \cdot f \cdot e(S, F) \end{aligned}$$

hence all checks of $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}, M, \sigma)$ pass.

Therefore SP-CHQS also satisfies evaluation correctness.

Finally it is an immediate corollary of Theorem 5.22 as well as Lemma 5.23 - 5.28, that no security reduction requires knowledge of messages m in $\mathbb{Z}_p$, but can be computed knowing group element $M = g_1^m \in \mathbb{G}_1$.

□

7.4.3 Combining SP-CHQS with Secret Sharing

We will now describe in detail the linear secret sharing based MPC we will combine with our Construction 7.33. Following the approach of Beaver [17] multiplications will be handled using preprocessing. This approach is used in modern MPC schemes (see e.g. [53]) as well as audit schemes for MPC (see e.g. [16, 91]. This combination of a secret sharing scheme with our FDC, obtained by applying transformation 7.5 to our homomorphic authenticator scheme from Construction 7.33 and our commitment scheme from construction 7.28, results in a verifiable computing scheme that provides both information-theoretic input and output privacy with respect to the verifier and information-theoretic input and output privacy with respect to the servers.

Construction 7.35.

$\mathsf{VKeyGen}(1^\lambda, \mathcal{P})$: *On input a security parameter λ and the description of a function f given as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, it runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}', \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, *as well as* $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. *It chooses* $y \xleftarrow{\$} \mathbb{Z}_p$ *uniformly at random. It sets* $Y = g_1^y$. *It sets* $\mathsf{sk} = (\mathsf{sk}', y, \mathcal{P})$, $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, Y, F)$ *and returns* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$.

Preprocessing*: The shareholders jointly generate M triples of t-reconstructing shares $s_k(a_j)$, $s_k(b_j)$, $s_k(c_j)$, reconstructing to to $(a_j, b_j], c_j) \subset \mathbb{F}_p$ for $j \in [M]$ such that for each the equation $a_j \cdot b_j = c_j$ holds (see, e.g. [52] for a detailed description of how to generate such shares).*

$\mathsf{ProbGen}(\mathsf{sk}, x)$*: On input the secret key* sk *and* $x = (m_1, \ldots, m_n, \Delta)$ *consisting of a tuple of n messages $m_i \in \mathbb{Z}_p$ for $i \in [n]$ and a dataset identifier $\Delta \in \{0,1\}^*$, it chooses $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random, as well as $r'_1, \ldots, r'_n \xleftarrow{\$} \mathbb{Z}_p$*

uniformly at random. It runs $\{s(m_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(m_i)$ *for all* $i \in [n]$. *as well as* $\{s(r_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(r_i)$ *for all* $i \in [n]$ *and* $\{s(r'_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(r'_i)$ *for all* $i \in [n]$. *It sets* $s_k(R_i) = g_2^{s_k(r_i)}$ *for all* $i \in [n], k \in [N]$ *and* $s_k(R'_i) = g_2^{s_k(r'_i)}$ *for all* $i \in [n], k \in [N]$.
It runs $A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, y, (r_i, r'_i), \Delta, \tau)$, *as well as* $Com_i \leftarrow \mathsf{PublicCommit}(\mathsf{sk}, m_i, y, (r_i, r'_i))$, *It chooses* $k^* \in [N]$. *This will identify the distinguished shareholder that will perform operations on authenticators.* k^* *can be chosen according to a clients preferences. It outputs the shares* $s_k(m_i)$, $s_k(R_i)$, $s_k(R'_i)$ *as well as* c_i *for* $i \in [n]$ *to shareholder* k *for* $k \in [N]$. *Additionally it outputs* $A_1, \ldots, A_n, Com_1, \ldots, Com_n$ *to shareholder* k^*. *It sets* $\rho_x = 0$ *and* $\sigma_x = (\Delta, \{A_i, Com_i, c_i, s_k(m_i), s_k(R_i), s_k(R'_i)\}_{i\in[n],k\in[N]})$. *It outputs* (σ_x, ρ_x).

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *On input an evaluation key* ek *and an encoded input* σ_x, *the algorithm parses* $\mathsf{ek} = (f, \tau_1, \ldots, \tau_n)$ *with* f *a multivariate poylnomial of degree 2, given by an arithmetic circuit and* $\sigma_x = (\Delta, \{A_i, Com_i, c_i, s_k(m_i), s_k(R_i), s_k(R'_i)\}_{i\in[n],k\in[N]})$. *Each shareholder* k *follows the circuit gate by gate:*

- Add_1**:** *On input shares* $(s_k(m_i), c_i, s_k(R_i), s_k(R'_i))$ *for* $i = 1, 2$, *it computes* $s_k(m) = s_k(m_1) + s_k(m_2)$, $c = c_1 + c_2$, $s_k(R) = s_k(R_1) \cdot s_k(R_2)$, *and* $s_k(R') = s_k(R'_1) \cdot s_k(R'_2)$.
- cMult_1**:** *On input shares* $(s_k(m'), c', s_k(S), s_k(S'))$ *and a constant* $\alpha \in \mathbb{Z}_p$, *it computes* $s_k(m) = \alpha \cdot s_k(m')$, $c = \alpha \cdot c'$, $s_k(R) = s_k(S)^\alpha$, *and* $s_k(R') = s_k(S')^\alpha$.
- Mult**:** *On input shares* $(s_k(m_i), c_i, s_k(R_i), s_k(R'_i))$ *for* $i = 1, 2$, *it takes multiplicative shares* $s_k(a), s_k(b), s_k(c)$ *and does the following:*

  Each shareholder k *computes the following*

  $$s_k(\delta) = s_k(m_1) - s_k(a)$$
  $$s_k(\epsilon) = s_k(m_2) - s_k(b)$$

  The shareholders jointly choose a subset $\mathcal{B} \subset [n]$ *of size* $|\mathcal{B}| \geq t$ *and run*

  $$\delta \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(\delta)\}_{k\in\mathcal{B}})$$
  $$\epsilon \leftarrow \mathsf{SReconstruct}(\mathcal{B}, \{s_k(\epsilon)\}_{k\in\mathcal{B}})$$

  Each shareholder k *computes* $s_k(m) = s_k(c) + \epsilon s_k(m_1) + \delta s_k(m_2) - \delta\epsilon$, *as well as* $s_k(R) = s_k(R_1)^{c_2}$, *and* $s_k(R') = s_k(R'_1)^{c_2} \cdot s_k(R'_2)$.
- Add_2**:** *On input shares* $(s_k(m_i), s_k(R_i), s_k(R'_i))$ *for* $i = 1, 2$, *it computes* $s_k(m) = s_k(m_1) + s_k(m_2)$, $s_k(R) = s_k(R_1) \cdot s_k(R_2)$, *and* $s_k(R') = s_k(R'_1) \cdot s_k(R'_2)$.
  Note, that such shares are the output of multiplication gates.
- cMult_2**:** *On input shares* $(s_k(m'), s_k(S), s_k(S'))$, *and a constant* $\alpha \in \mathbb{Z}_p$, *it computes* $s_k(m) = \alpha \cdot s_k(m')$, $s_k(R) = s_k(S)^\alpha$, *and* $s_k(R') = s_k(S')^\alpha$.

Shift*: On input shares $(s_k(m), c, s_k(R), s_k(R'))$, it returns $(s_k(m), s_k(R), s_k(R'))$.*

The final outputs are $s_k(m^*), s_k(R), s_k(R')$.

Additionally k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$, as well as $Com^* \leftarrow \mathsf{CEval}(f, C_1, \ldots, C_n)$. It sets $\sigma_y = (\Delta, A^*, C^*, \{s_k(m^*), s_k(r^*)\}_{k\in[N]})$ and outputs σ_y.

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: *On input a verification key vk, a decoding value ρ_x and an encoded value σ_y, it parses $\mathsf{vk} = (\mathsf{pk}, F)$, $\rho_x = 0$, $\sigma_y = (\Delta, A^*, C^*, \{s_k(m^*), s_k(R), s_k(R')\}_{k\in[N]})$. It chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| \geq t$. It creates the reconstruction vector $(w_1, \ldots, w_t)$ derived from $\mathcal{B}$ (see Sec. 2.4 for details). It computes $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$, as well as $R = \prod_{k\in\mathcal{B}} s_k(R)^{w_k}$ and $R' = \prod_{k\in\mathcal{B}} s_k(R')^{w_k}$. Finally, it runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, Com^*, F, \Delta)$. If $b = 0$ it outputs $\perp$ else it runs $b' \leftarrow \mathsf{PublicDecommit}(Com^*, m^*, Y, R, R')$. If $b' = 0$ it outputs $\perp$, else it outputs m^*.*

We now look at the basic properties of this construction. A first requirement is obviously correctness, showing that any honest execution of the algorithm leads to verifiers accepting a correct result.

Proposition 7.36. *Construction 7.35 is a correct verifiable computing scheme in the sense of Def. 2.30*

Proof. Let f be an arbitrary circuit of multiplicative depth 2, $x = (m_1, \ldots, m_m, \Delta)$ be an arbitrary input. Let f be described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Let $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$, $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$, $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$. as well as $y = f(m_1, \ldots, m_n)$.

We follow the circuit gate by gate, and show that correctness holds at every step.

$\mathsf{Add_1}$: We have $s_k(m) = s_k(m_1) + s_k(m_2) = s_k(m_1 + m_2)$, $c = c_1 + c_2$, $s_k(R) = s_k(R_1) \cdot s_k(R_2) = s_k(R_1 \cdot R_2)$, and $s_k(R') = s_k(R'_1) \cdot s_k(R'_2) = s_k(R'_1 \cdot R'_2)$. This is correct as a corollary of Proposition 7.32.

$\mathsf{cMult_1}$: We have $s_k(m) = \alpha \cdot s_k(m')$, $c = \alpha \cdot c'$, $s_k(R) = s_k(S)^\alpha$, and $s_k(R') = s_k(S')^\alpha$. This is correct as a corollary of Proposition 7.32.

Mult: We have $s_k(m) = s_k(m_1 \cdot m_2)$ as after the preprocessing stage the shareholders hold t-reconstructing shares $s_k(a)$, $s_k(b)$, $s_k(c)$, such that $a \cdot b = c$ holds. We have

$$
\begin{aligned}
&s_k(m) = s_k(c) + \epsilon s_k(m_1) + \delta s_k(m_2) - \delta\epsilon \\
&= s_k(ab) + (m_2 - b)s_k(m_1) + (m_1 - a)s_k(m_2) - (m_1 - a)(m_2 - b) \\
&= s_k(ab) + s_k(m_1 m_2 - m_1 b) + s_k(m_1 m_2 - m_2 a) - m_1 m_2 + m_1 b + m_2 a - ab \\
&= s_k(m_1 m_2 + ab - ab + m_1 b - m_1 b + m_2 a - m_2 a) \\
&= s_k(m_1 m_2)
\end{aligned}
$$

Furthermore we have $s_k(R) = s_k(R_1)^{c_2}$, and $s_k(R') = s_k(R'_1)^{c_2} \cdot s_k(R'_2)$. This is correct as a corollary of Proposition 7.32.

Add_2**:** We have $s_k(m) = s_k(m_1) + s_k(m_2) = s_k(m_1 + m_2)$, $s_k(R) = s_k(R_1) \cdot s_k(R_2) = s_k(R_1 \cdot R_2)$, and $s_k(R') = s_k(R'_1) \cdot s_k(R'_2) = s_k(R'_1 \cdot R'_2)$. This is correct as a corollary of Proposition 7.32.

cMult_2**:** We have $s_k(m) = \alpha \cdot s_k(m')$, $s_k(R) = s_k(S)^\alpha$, and $s_k(R') = s_k(S')^\alpha$. This is correct as a corollary of Proposition 7.32.

Shift**:** This is trivially correct as a corollary of Proposition 7.32.

Additionally k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \ldots, A_n)$ as well as $Com^* \leftarrow \mathsf{Eval}(f, Com_1, \ldots, Com_n)$. We parse $\sigma_y = (\Delta, A^*, Com^*, \{s_k(m^*), s_k(R), s_k(R')\}_{k\in[N]})$, as the output of the final gate. We set $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$, as well as $R = \prod_{k\in\mathcal{B}} s_k(R)^{w_k}$ and $R' = \prod_{k\in\mathcal{B}} s_k(R')^{w_k}$. We have $\mathsf{PublicDecommit}(Com^*, Y, m^*, R, R') = 1$ as a corollary of Proposition 7.32. By the correctness of our FDC scheme (see Theorem 4.14) we have therefore $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) = 1$.

□

Next we consider the case of third party verifiers and show that this construction is even publicly verifiable.

Proposition 7.37. *Construction 4.27 is a publicly verifiable computing scheme.*

Proof. Note, that $\rho_x = 0$ by definition. Obviously this does not need to be kept secret. Since we have $\mathsf{vk} = (\mathsf{pk}, F)$, where pk is the public key of the FDC scheme (see 4.13) and F is a function commitment. Both values are public. □

Now we formally show that this combination of our FDC with the linear secret sharing scheme described above does indeed lead to secure verifiable computing scheme. In order to provide this security reduction we will define a sequence of games, each dealing with a specific type of forgery. In a series of Lemmata we will bound the distance between those games and finally show that this leads to a negligible advantage for the adversary to produce an incorrect result, that would falsely be accepted as correct.

Theorem 7.38. *Construction 7.35 is an adaptively secure verifiable computing scheme in the sense of Def. 2.33.*

Proof. To prove Theorem 7.38, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. the game outputs '1' only with negligible probability. Following the notation of [37], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns '1'. We use flag values bad_i, initially set to false. If at the end of the game any of these flags is set to true, the game simply outputs '0'. Let Bad_i denote the event that bad_i is set to true during game i.

Game 1 is the security experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda]$ (see Def. 2.33) between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Game 2 is is defined as Game 1, except for the following change: Whenever $\mathcal{A}$ returns a forgery σ_y, $\mathcal{C}$ parses $\sigma_y = (\Delta, A^*, C^*, \{s_k(m^*), s_k(S), s_k(S')\}_{k\in[N]})$. It runs $\hat{\sigma}_y = (\Delta, \hat{A}, \hat{C}, \{s_k(\hat{m}), s_k(R), s_k(R')\}_{k\in[N]}) \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$ honestly. Then, it chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| = t$. It creates the reconstruction vector $(w_1, \ldots, w_t)$ derived from $\mathcal{B}$ (see Sec. 2.4 for details). It computes $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$, as well as $S = \prod_{k\in\mathcal{B}} s_k(S)^{w_k}$ and $S' = \prod_{k\in\mathcal{B}} s_k(S')^{w_k}$. It also computes $\hat{m} = \sum_{k\in\mathcal{B}} w_k s_k(\hat{m})$, as well as $R = \prod_{k\in\mathcal{B}} s_k(R)^{w_k}$ and $R' = \prod_{k\in\mathcal{B}} s_k(R')^{w_k}$.
It runs $b' \leftarrow \mathsf{PublicDecommit}(\hat{C}, m^*, Y, (S, S'))$. If $b' = 1$ it sets $\mathsf{bad}_2 = \mathsf{true}$.
In Lemma 7.39, we show that any adversary $\mathcal{A}$ for which Bad_2 occurs implies an adversary breaking either the aug-DDH assumption (see Def. 7.29) or the dpa_1 assumption (see Def. 2.40).
Finally in Lemma 7.40 we show that that any adversary $\mathcal{A}$ that wins Game 2 implies an adversary winning experiment $\mathbf{EXP}_{\mathcal{A},\mathrm{FDC}}^{UF-CMA}$ (see Def. 4.9).

□

Lemma 7.39. *We have* $\Pr[\mathsf{Bad}_2] \leq \mathsf{Adv}_{\mathcal{S}}^{\mathsf{dpa}_1}(\lambda) + \mathsf{Adv}_{\mathcal{S}}^{aug-DDH}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 2 such that Bad_2 occurs. We then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to break either the augmented DDH assumption (see Def. 7.29) or the dba_1 assumption (see Def. 2.40).

Analogously to Prop. 7.30, we distinguish between two cases:

Case 1: We have $R = S$ in Game 2.
Case 2: We have $R \neq S$ in Game 2.

We first consider case 1:
$\mathcal{S}$ takes as input $\mathsf{bgp}, g_1^x, g_2^{\frac{1}{x}}, g_1^y, g_2^z$.

Setup $\mathcal{S}$ chooses an arbitrary quadratic function f described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$.
It chooses $y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $Y = g_1^y$.
It sets $\mathsf{bgp}' = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1^x, g_2, e)$, It sets $H = g_1$. Additionally, it fixes a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Ver}_{\mathsf{Sig}})$ with signature space $\Sigma \subset \mathsf{bgp}$ and a pseudorandom function $\Phi : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It outputs the public parameters $\mathsf{pp} = (\lambda, n, \mathsf{bgp}', H, \mathsf{Sig}, \Phi)$.
Then, it chooses $x \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It sets $h_2 = g_2^x$. It samples $t_{\tau_i}, k_{\tau_i} \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for all $i \in [n]$ and sets $F_{\tau_i} = g_2^{t_{\tau_i}}$, as well as $f_{\tau_i} = g_t^{y t_{\tau_i}}$, $f_{\tau_i,\tau_j} = g_t^{t_{\tau_i} k_{\tau_j}}$, for all $i, j \in [n]$. Additionally the algorithm

chooses a random seed $K \xleftarrow{\$} \mathcal{K}$ for the pseudorandom function Φ. It computes keys for the regular signature scheme $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. It sets $\mathsf{pk} = (\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, h_2, \{F_{\tau_i}, f_{\tau_i}\}_{i=1}^n, \{f_{\tau_i,\tau_j}\}_{i,j=1}^n)$. It honestly runs $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. It chooses $y \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $Y = g_1^y$. It $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, Y, F)$ and returns $(\mathsf{ek}, \mathsf{pk})$ to the adversary $\mathcal{A}$.

Queries When $\mathcal{A}$ queries $x = (m_1, \ldots, m_n, \Delta)$ $\mathcal{S}$ does the following:

It chooses $r_i, r_i' \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for $i \in [n]$. and sets $c_i = m_i + yr_i$ as well as $C_i = (g_1^x)^{m_i} \cdot (g_1^x)^{r_i} \cdot g_1^{yr_i'}$ for $i \in [n]$. Furthermore it sets $R_i = \left(g_2^{\frac{1}{x}}\right)^{r_i}$, $R' = g_2^{r_i'}$ and $d_i = (R_i, R_i')$ for $i \in [n]$. Note that this is perfectly indistinguishable from an honest execution of $\mathsf{PublicCommit}(\mathsf{sk}, m_i, y, (r_i, r_i'))$. It then performs the rest of $\mathsf{ProbGen}$ honestly. This is perfectly indistinguishable from an honest execution of the algorithm to $\mathcal{A}$.

Forgery The adversary $\mathcal{A}$ returns σ_y^*. Simulator $\mathcal{S}$ parses $\sigma_y^* = (\Delta,\ A^*,\ Com^*,\ \{s_k(m^*),\ s_k(R),\ s_k(R')\}_{k\in[N]})$. It chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| = t$ to reconstruct the message and decommitments, i.e. it runs
$m^* \leftarrow \mathsf{SReconstruct}(\{s_k(m^*)\}_{k\in\mathcal{B}})$, $R \leftarrow \mathsf{SReconstruct}(\{s_k(R)\}_{k\in\mathcal{B}})$, and $R' \leftarrow \mathsf{SReconstruct}(\{s_k(R')\}_{k\in\mathcal{B}})$.
$\mathcal{S}$ runs $\hat{\sigma}_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$ honestly and parses $\hat{\sigma}_y = (\Delta,\ \hat{A},\ \hat{Com},\ \{s_k(\hat{m}),\ s_k(S),\ s_k(S')\}_{k\in[N]})$ Since we have $\mathsf{Bad}_2 = \mathsf{true}$ we have $\hat{Com} = Com^*$. We parse $Com^* = (C^*, Y)$. By assumption we have

$$e\left(C^*, g_2\right) = e\left(g_1^{m^*}, g_2\right) \cdot e\left(g_1^x, R\right) \cdot e\left(g_1^y, R'\right) = e\left(g_1^{m^*}, g_2\right) \cdot e\left(H, R \cdot (R')^{\frac{x}{y}}\right)$$

as well as

$$e\left(C^*, g_2\right) = e\left(g_1^{\hat{m}}, g_2\right) \cdot e\left(g_1^x, S\right) \cdot e\left(g_1^y, S'\right) = e\left(g_1^{m^*}, g_2\right) \cdot e\left(H, S \cdot (S')^{\frac{x}{y}}\right)$$

Dividing those two equations and using the fact that $R = S$ (since we consider case 1) yields

$$1 = e\left(\left(g_1^x\right)^{m-m'}, g_2\right) \cdot e\left(Y, \frac{R'}{S'}\right)$$

or equivalently

$$e\left(\left(g_1^x\right)^{m'-m}, g_2\right) \cdot e\left(g_1, \frac{R'^{\,y}}{S'}\right)$$

Since we know that $m \neq m'$ we have $g_2^x = \left(\frac{R'}{S'}\right)^{\frac{y}{m'-m}}$.
We now have $z = xy$ if and only if

$$e\left(g_1, g_2^z\right) = e\left(g_1^y, \left(\frac{R'}{S'}\right)^{\frac{y}{m'-m}}\right)$$

holds.

This shows that case 1 implies a solver for the aug-DDH problem.

We now consider case 2:

$\mathcal{S}$ takes as input $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_1^x, g_2, e)$. It seeks to compute $G, G_x \in \mathbb{G}_2$ such that $1 = e(g_1, G) \cdot e(g_1^x, G_x)$.

Setup $\mathcal{S}$ chooses an arbitrary quadratic function f described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$.
It sets $H = g_1^x$ (taken from the input) and runs the rest of $\mathsf{VKeyGen}$ honestly returning $(\mathsf{ek}, \mathsf{vk})$ to the adversary $\mathcal{A}$.

Queries When $\mathcal{A}$ queries $x = (m_1, \ldots, m_n, \Delta)$ $\mathcal{S}$ does the following:
It chooses $r_i, r_i' \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for $i \in [n]$. and sets $c_i = m_i + yr_i$ as well as
$C_i = g_1^{m_i} \cdot H^{r_i} \cdot g_1^{yr_i'}$ for $i \in [n]$. Furthermore it sets $R_i = g_2^{r_i}$, $R' = g_2^{r_i'}$ and $d_i = (R_i, R_i')$ for $i \in [n]$. Note that this is perfectly indistinguishable from an honest execution of $\mathsf{PublicCommit}(\mathsf{sk}, m_i, y, (r_i, r_i'))$. It then performs the rest of $\mathsf{ProbGen}$ honestly. This is perfectly indistinguishable from an honest execution of the algorithm to $\mathcal{A}$.

Forgery The adversary $\mathcal{A}$ returns σ_y^*. $\mathcal{S}$ parses $\sigma_y^* = (\Delta, A^*, Com^*, \{s_k(m^*), s_k(R), s_k(R')\}_{k\in[N]})$. It chooses a subset $\mathcal{B} \subset [N]$ with $|\mathcal{B}| = t$ and runs $m^* \leftarrow \mathsf{SReconstruct}(\{s_k(m^*)\}_{k\in\mathcal{B}})$, $R \leftarrow \mathsf{SReconstruct}(\{s_k(R)\}_{k\in\mathcal{B}})$, and $R' \leftarrow \mathsf{SReconstruct}(\{s_k(R')\}_{k\in\mathcal{B}})$.
$\mathcal{S}$ runs $\hat{\sigma}_y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$ honestly and parses $\hat{\sigma}_y = (\Delta, \hat{A}, \hat{Com}, \{s_k(\hat{m}), s_k(S), s_k(S')\}_{k\in[N]})$ Since we have $\mathsf{Bad}_2 = \mathsf{true}$ we have $\hat{Com} = Com^*$. We parse $Com^* = (C^*, Y)$.
By the correctness of our algorithm we have

$$e(C, g_2) = e(g_1^m, g_2) \cdot e(H, R) \cdot e(Y, R')$$

If $\mathcal{A}$ wins the security game we have

$$e(C, g_2) = e\left(g_1^{m'}, g_2\right) \cdot e(H, S) \cdot e(Y, S').$$

Dividing the equations yields

$$1 = e\left(g_1^{m-m'}, g_2\right) \cdot e\left(H, \frac{R}{S}\right) \cdot e\left(Y, \frac{R'}{S'}\right)$$

or equivalently

$$1 = e\left(g_1, g_2^{m-m'} \cdot \frac{R'^{\,y}}{S'}\right) \cdot e\left(g_1^x, \frac{R}{S}\right)$$

We know that $R \neq S$ (case 2). Therefore $G = g_2^{m-m'} \cdot \frac{R'^{\,y}}{S'}$ and $G_x = \frac{R}{S}$ is a valid solution for the DBP_1 problem.

□

Lemma 7.40. *For every PPT adversary $\mathcal{A}$ running Game 2, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_2] \leq \mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{CHQS}}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during the security experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathsf{VC}, f, \lambda]$ (see Def. 2.33), we then show how a simulator $\mathcal{S}$ can use $\mathcal{A}$ to win the security experiment $\mathbf{EXP}_{\mathcal{A},\mathrm{FDC}}^{UF-CMA}$ (see Def. 4.9).

Setup Simulator $\mathcal{S}$ runs $\mathsf{pp} \leftarrow \mathsf{HSetup}(1^\lambda)$ and outputs pp. It chooses an arbitrary quadratic function f described as a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$. Let be an , $x = (m_1, \ldots, m_m, \Delta)$ be an arbitrary input. Let $(\mathsf{sk}, \mathsf{ek}, \mathsf{pk}) \leftarrow \mathsf{VKeyGen}(1^\lambda, \mathcal{P})$.

Key Generation Simulator $\mathcal{S}$ runs $(\mathsf{sk}', \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. By construction we have $\mathsf{ek} = 0$. Furthermore it runs $F \leftarrow \mathsf{PublicCommit}(\mathsf{pk}, \mathcal{P})$. It sets $\mathsf{sk} = (\mathsf{sk}', \mathcal{P})$, $\mathsf{ek} = \mathcal{P}$, $\mathsf{vk} = (\mathsf{pk}, F)$ and outputs $(\mathsf{ek}, \mathsf{pk})$ to the adversary $\mathcal{A}$.

Queries When $\mathcal{A}$ queries $x = (m_1, \ldots, m_n, \Delta)$ $\mathcal{S}$ does the following. It chooses $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random, as well as $r'_1, \ldots, r'_n \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. It computes $Com_i \leftarrow \mathsf{PublicCommit}(m_i, y, r_i, r'_i)$ for all $i \in [n]$, and queries for $(\Delta, \tau_i, m_i, y, r_i, r'_i)$ for $i \in [n]$, receiving A_i. It runs $\{s_k(m_i\}_{k\in[N]} \leftarrow \mathsf{SShare}(m_i)$ for $i \in n$ as well as as well as $\{s(r_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(r_i)$ for all $i \in [n]$ and $\{s(r'_i)\}_{k\in[N]} \leftarrow \mathsf{SShare}(r'_i)$ for all $i \in [n]$. It sets $s_k(R_i) = g_2^{s_k(r_i)}$ for all $i \in [n], k \in [N]$ and $s_k(R'_i) = g_2^{s_k(r'_i)}$ for all $i \in [n], k \in [N]$. It outputs $\sigma_x = (\Delta, \{A_i, Com_i, s_k(m_i), s_k(R_i), s_k(R'_i)\}_{i\in[n],k\in[N]})$. Note that this is the identical response to an honest evaluation of $\mathsf{ProbGen}$.

Forgery $\mathcal{A}$ returns σ_y^*.

$\mathcal{S}$ parses $\sigma_y^* = (\Delta,\ A^*,\ Com^*,\ \{s_k(m^*),\ s_k(R),\ s_k(R')\}_{k\in[N]})$. It sets $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}, \Delta^*)$ It runs $\hat{Com} \leftarrow \mathsf{CEval}(f, Com_1, \ldots, Com_n)$ Since we have $\mathsf{Bad}_2 = \mathsf{false}$ we have $Com^* \neq \hat{Com}$. Therefore $(\mathcal{P}^*_{\Delta^*}, A^*, C^*)$ is a type 2 forgery as defined in Def. 4.8 if and only if $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y) = y$, and $y \neq f(x)$.

Thus the claim follows from Proposition 7.10 and Theorem 5.22. □

Furthermore we can show that our construction preserves efficient verification. After a one time function-dependent preprocessing verification can indeed be faster than a computation of the function itself.

Proposition 7.41. *Construction 7.35 is a verifiable computing scheme that achieves amortized efficiency in the sense of Def. 2.35 for all f with super linear runtime in n.*

Proof. As a corollary of Propositions 5.20 and 7.12 an evaluation of $\mathsf{FunctionVerify}$ has runtime $\mathcal{O}(n)$. Therefore, our construction achieves amortized efficiency for suitably large n. □

Finally we show that our verifiable computing scheme achieves complete information theoretic privacy. Over the following four propositions we prove that it offers information-theoretic input privacy with respect to the servers, information-theoretic output privacy with respect to the servers, information-theoretic input privacy with respect to the verifier, and information-theoretic output privacy with respect to the verifier which have each been defined in Section 3.1.

Proposition 7.42. *Construction 7.35 achieves information-theoretic input privacy with respect to the servers in the sense of Def. 3.1 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:**

Let f be a multivariate polynomial of degree 2 and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

Let $m_1, \ldots m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages.

Setting $x_0 = (m_1, \ldots m_n, \Delta)$ and $x_1 = (m'_1, \ldots m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$.

The adversary $\mathcal{A}$ chooses a subset $\mathcal{B} \subset [N]$ of size $|\mathcal{B}| = t-1$.

We assume $k^* \in \mathcal{B}$. If the adversary does not corrupt k^* the claim immediately follows from the hiding property of Shamir secret sharing [97].

Thus the adversary obtains and seeks to distinguish $(x_0,\, x_1,\, \Delta,\, \{A_i,\, s_k(m_i[j]),\, s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$ and $(x_0,\, x_1,\, \Delta,\, \{A'_i,\, s_k(m'_i[j]),\, s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.

By the hiding property of Shamir secret sharing
$(x_0, x_1, \Delta, \{A_i, s_k(m_i[j]), s_k(r_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$
is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A_i, R_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
Obviously this is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A_i, R'_{ijk} \mid A_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m_i, r_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$,
as this is just another sampling of randomness. By the hiding property of the FDC (see Proposition 7.31), this is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A'_i, R'_{ijk} \mid A'_i \leftarrow \mathsf{PrivateCommit}(\mathsf{sk}, m'_i, r'_i, \Delta, \tau_i), R'_{ijk} \xleftarrow{\$} \mathbb{Z}_p\}_{i\in[n],j\in[T],k\in\mathcal{B}})$.
Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from
$(x_0, x_1, \Delta, \{A'_i, s_k(m'_i[j]), s_k(r'_i)\}_{i\in[n],j\in[T],k\in\mathcal{B}})$. This completes the proof.

□

Proposition 7.43. *Construction 7.35 achieves information-theoretic output privacy with respect to the servers in the sense of Def. 3.2 against an adversary corrupting at most $t-1$ shareholders.*

Proof. **Setup:** This setup is identical to Proposition 7.42.

Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y_0 = \sum_{i=1}^{n} f_i m_i$, $y_1 = \sum_{i=1}^{n} f_i m'_i$, $r = \sum_{i=1}^{n} f_i r_i$, $r' = \sum_{i=1}^{n} f_i r'_i$.
We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y_0), s_k(r)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y_1), s_k(r')\}_{k\in[N]})$. Thus the adversary obtains and seeks to distinguish $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ and $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$.
By the hiding property of Shamir secret sharing $(y_0, y_1, \Delta, A, \{s_k(y_0), s_k(r)\}_{k\in\mathcal{B}})$ is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R_k, S_k \mid R_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Obviously this is perfectly indistinguishable from $(y_0, y_1, \Delta, \{A, R'_k, S'_k \mid R'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. as this is just another sampling of randomness. By the hiding property of the FDC (see Proposition 7.31), this is perfectly indistinguishable from
$(y_0, y_1, \Delta, \{A', R'_k, S'_k \mid R'_k \xleftarrow{\$} \mathbb{G}_1\}_{k\in\mathcal{B}})$. Again by the hiding property of Shamir secret sharing this is perfectly indistinguishable from $(y_0, y_1, \Delta, A', \{s_k(y_1), s_k(r')\}_{k\in\mathcal{B}})$. This completes the proof.

□

Proposition 7.44. *Construction 7.35 achieves information-theoretic input privacy with respect to the verifier in the sense of Def. 3.3.*

Proof. Let f be a multivariate polynomial of degree 2, $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

Let $m_1, \ldots m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \ldots m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages. such that $f(m_1, \ldots, m_n) = f(m'_1, \ldots, m'_n)$.

Setting $x_0 = (m_1, \ldots m_n, \Delta)$ and $x_1 = (m'_1, \ldots m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$.

Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0,1\}$ and $y = f(m_1, \ldots, m_n)$,

Let $d_i = (c_i, R_i, R'_i)$ and $d'_i = (c'_i, S_i, S'_i)$ for all $i \in [n]$ and $d = f'(d_1, \ldots, d_n)$, $d' = f'(d'_1, \ldots, d'_n)$ following $\mathsf{Compute}$. By assumption we have $y = f(m'_1, \ldots, m'_n)$.
We parse $\sigma_{y_0} = (\Delta, A, \{s_k(y), s_k(d)\}_{k\in[N]})$ and $\sigma_{y_1} = (\Delta, A', \{s_k(y), s_k(d')\}_{k\in[N]})$.
Thus the adversary obtains and seeks to distinguish $(x_0, x_1, \Delta, A, \{s_k(y), s_k(d)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(d')\}_{k\in\mathcal{B}})$.

Note that an adversary $\mathcal{A}$ that can distinguish $(x_0, x_1, \Delta, A, \{s_k(y), s_k(d)\}_{k\in\mathcal{B}})$ and $(x_0, x_1, \Delta, A', \{s_k(y), s_k(d')\}_{k\in\mathcal{B}})$ immediately implies an adversary $\mathcal{A}'$ that can distinguish (x_0, x_1, Δ, y, d) and $(x_0, x_1, \Delta, y, d')$. Since both d and d' are distributed uniformly at random, these are perfectly indistinguishable.

□

Proposition 7.45. *Construction 7.35 achieves information-theoretic output privacy with respect to the verifier in the sense of Def. 3.4.*

Proof. Let f be a linear function given by its coefficient vector $(f_1, \ldots f_n)$ and $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and $\Delta \leftarrow \{0,1\}^*$ a dataset identifier.

We first show correctness in the sense of Def. 3.4. We provide the three additional algorithms.

$\mathsf{HideCompute}(\mathsf{ek}, \sigma_x)$: The computation algorithm takes the evaluation key ek and the encoded input σ_x. k^* runs $A^* \leftarrow \mathsf{Eval}(f, A_1, \dots A_n)$ as well as $C^* \leftarrow \mathsf{ComEval}(Com_1, \dots, Com_n)$ It sets $\tilde{\sigma_y} = (\Delta, A^*, C^*)$ and outputs the encoded version $\tilde{\sigma_y}$.

$\mathsf{HideVerify}(\mathsf{vk}, \tilde{\sigma_y})$: On input a verification key $\mathsf{vk} = (\mathsf{pk}, F)$ and an encoded value $\tilde{\sigma_y}$, it parses $\tilde{\sigma_y} = (\Delta, A^*, C^*\}_{k\in[N]})$. Then, it runs $b \leftarrow \mathsf{FunctionVerify}(\mathsf{pk}, A^*, C^*, F, \Delta)$. If $b = 0$ it outputs $\perp$, else it sets $\hat{\sigma}_y = (\Delta, A^*, C^*)$ and outputs $\hat{\sigma}_y$.

$\mathsf{Decode}(\mathsf{vk}, \rho_x, \hat{\sigma}_y, \sigma_y)$: It takes as input the verification key vk, a decoding value $\rho_x = 0$, and encoded values $\hat{\sigma}_y, \sigma_y$. It parses $\sigma_y = (\Delta, A^*, \{s_k(m^*), s_k(d^*)\}_{k\in[N]})$, $\hat{\sigma}_y = (\Delta, A^*, C^*)$. It computes $m^* = \sum_{k\in\mathcal{B}} w_k s_k(m^*)$. Then it parses $s_k(d^*) = (s_k(R^*), s_k(S^*))$, and computes $R^* = \prod_{k\in\mathcal{B}} s_k(R^*)^{w_k}$, $S^* = \prod_{k\in\mathcal{B}} s_k(S^*)^{w_k}$. Finally, it sets $d^* = (R^*, S^*)$ and runs $b \leftarrow \mathsf{PublicDecommit}(C^*, m^*, d^*)$. If $b = 0$ it outputs $\perp$ else it returns m^*.

Now we show privacy in the sense of Def. 3.4.
Let $m_1, \dots, m_n \leftarrow \mathbb{Z}_p^T$ and $m'_1, \dots, m'_n \leftarrow \mathbb{Z}_p^T$ be two tuples of messages.
Setting $x_0 = (m_1, \dots m_n, \Delta)$ and $x_1 = (m'_1, \dots m'_n, \Delta)$, let $\sigma_{x_i} \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$.
Let $\sigma_{y_i} \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_{x_i})$ for $i \in \{0, 1\}$ and $y_0 = f(m_1, \dots, m_n)$, $y_1 = f(m'_1, \dots, m'_n)$.
We parse $\sigma_{y_0} = (\Delta, A, C)$ and $\sigma_{y_1} = (\Delta, A', C)$.
Note that an adversary $\mathcal{A}$ seeks to distinguish (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$. However, (y_0, y_1, Δ, A, C) and $(y_0, y_1, \Delta, A', C')$ are perfectly indistinguishable by the hiding property of the FDC (see Proposition 7.31).

□

8 Conclusion

Summary of our contributions. This thesis provides the first solutions for verifiable computing providing complete information-theoretic privacy. We recall that in the setting of publicly verifiable computations we consider input-output privacy with respect to both server and verifier. This allows a data owner to keep its sensitive data private while still benefiting from outsourced computations. Within this thesis we presented the first schemes to achieve such complete privacy in an information-theoretic sense, allowing verifiable computation to be used even for highly sensitive data.

To this end, we introduced the framework of function-dependent commitments as a building block to achieving even information-theoretic privacy with respect to both verifier and server when using verifiable computing. We then focused on homomorphic authenticators fine-tailored to specific computations that already achieve information-theoretic input privacy with respect to the verifier. In particular we considered the setting of multi-key homomorphic authenticators, where multiple data owners provide data, authenticated under multiple keys. Furthermore, we considered multivariate polynomials of degree two. Afterwards we focused on adding computational privacy to linearly homomorphic authenticators under the LAEPuV framework (see Chapter 6), thereby constructing the first concrete instantiations without false negatives. This resulted in two schemes (Con. 6.9 and Con. 6.25) with computational privacy. Finally, in Chapter 7 we looked at a transformation for certain homomorphic authenticators turning them into FDCs and thereby adding information-theoretic privacy. We then applied this transformation to schemes developed in this thesis. This, along with a direct construction of an FDC in chapter 4, resulted in three schemes with complete information-theoretic privacy, one fine-tailored towards linear functions, another targeted at quadratic functions, and a final one, aimed at linear functions evaluated over inputs from multiple data owners.

In table 8.1 we compare our novel schemes to existing authenticator-based verifiable computing schemes (note that we use the abbreviations introduced in table 3.1). Here we can see that our constructions are the first to provide complete privacy and even complete information-theoretic privacy. Furthermore, our schemes

can be built from bilinear maps (or the RSA assumption), avoiding primitives like SNARKs, that are only known to exist under strong, non-falsifiable assumptions.

Scheme	Function Class	$\mathcal{A}$	PrS	Primitives	E	VER	PrV
[12]	Poly. of Degree 2	S	×	Bilinear Maps	A	×	×
[104]	Poly. of Fixed Degree	S	×	Multilinear Maps	A	×	×
[58]	Poly of Degree 2	S	I	Bilinear Maps	A	×	×
[37]	Linear	S	×	Bilinear Maps	A	✓	I(inf.)
[41]	Poly of Fixed Degree	S	×	Multilinear Maps	A	✓	NA
[36]	Poly of Fixed Degree	S	×	RSA	A	✓	NA
[75]	D	D	I/O	HE/HEA	D	D	D
[59]	Arithm. Circuits	S	×	Lattices	A	✓	×
[76]	D	S	×	SNARKs	A	✓	D
Con. 6.9	Linear	S	I/O	RSA	×	✓	I/O
Con. 6.25	Linear	S	I/O	Bilinear Maps	A	✓	I(inf)/O
Con. 4.27	Linear	S	I/O(inf.)	Bilinear Maps	A	✓	I/O (inf.)
Con. 7.35	Poly of Degree 2	S	I/O(inf.)	Bilinear Maps	A	✓	I/O (inf.)
Con. 7.19	Linear (Multi-Key)	S	I/O(inf.)	Bilinear Maps	A	✓	I/O (inf.)

Table 8.1: New authenticator-based verifiable computing schemes. Properties: adversary ($\mathcal{A}$), privacy w.r.t server (**PrS**), efficiency (**E**), public verifiability (**VER**), privacy w.r.t verifier (**PrV**)

Directions for future work. Many of the schemes presented in this work make use of cryptographic bilinear maps. One natural generalization of bilinear maps are multilinear maps. Assuming secure instantiations of such multilinear maps exist, applying our transformation to multilinear map based homomorphic authenticators would lead to FDCs supporting generic arithmetic circuits of bounded depth. This obviously allows for verifiable computations of a broader class of functions. In this work we did not consider constructions based on non-falsifiable assumptions, so in particular we did not consider homomorphic authenticators derived from SNARKs. Building FDCs for these constructions is another research challenge that could allow for FDCs supporting a broader class of functions, without the need for new cryptographic primitives, but at the cost of stronger assumptions. Finally, we note that our Construction 7.5 was formulated using the existing framework of structure-preserving authenticators. It does however not explicitly require all the properties of structure-preserving authenticators. In fact, our transformation can be applied to any homomorphic commitment scheme where the commitment space is contained within the message space of a homomorphic authenticator scheme. Thus constructing novel homomorphic commitment and authenticator schemes can lead to new, possibly even more powerful FDC schemes. For instance, Gorbunov et al. [70] presented a lattice-based fully homomorphic commitment scheme, where the commitments are matrices over finite fields and homomorphic evaluations involve

both matrix multiplications and additions. Finding a homomorphic authenticator scheme supporting this specific operation would allow us to derive an FDC beyond the group setting.

Bibliography

[1] M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 4–24. Springer, 2012.

[2] M. Abe, K. Haralambiev, and M. Ohkubo. Signing on elements in bilinear groups for modular protocol design. *IACR Cryptology ePrint Archive*, 2010:133, 2010.

[3] S. Agrawal and D. Boneh. Homomorphic macs: Mac-based integrity for network coding. In *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 292–305, 2009.

[4] S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman. Preventing pollution attacks in multi-source network coding. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2010.

[5] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010.

[6] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[7] N. Attrapadung and B. Libert. Homomorphic network coding signatures in the standard model. In *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 17–34. Springer, 2011.

[8] N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 367–385. Springer, 2012.

[9] N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *Public Key Cryptography*,

volume 7778 of *Lecture Notes in Computer Science*, pages 386–404. Springer, 2013.

[10] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31. ACM, 1991.

[11] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *IEEE Symposium on Security and Privacy*, pages 271–286. IEEE Computer Society, 2015.

[12] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM Conference on Computer and Communications Security*, pages 863–874. ACM, 2013.

[13] M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. In *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 296–312. Springer, 2012.

[14] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 1997.

[15] P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.

[16] C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In *SCN*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2014.

[17] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[18] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[19] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796. USENIX Association, 2014.

[20] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

[21] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[22] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[23] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.

[24] D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2009.

[25] D. Boneh, K. Rubin, and A. Silverberg. Finding composite order ordinary elliptic curves using the cocks-pinch method. *Journal of Number Theory*, 131(5):832–841, 2011. Cryptology ePrint Archive, Report 2009/533.

[26] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[27] S. Bowe. BLS12-381: New zk-snark elliptic curve construction. Zcash Company, Mar. 11, 2017, `https://z.cash/blog/new-snark-curve` (visited on 02/19/2019).

[28] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519. Springer, 2014.

[29] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.

[30] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[31] B. Braun, A. J. Feldman, Z. Ren, S. T. V. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *SOSP*, pages 341–357. ACM, 2013.

[32] J. Braun, J. A. Buchmann, D. Demirel, M. Geihs, M. Fujiwara, S. Moriai, M. Sasaki, and A. Waseda. LINCOS: A storage system providing long-term integrity, authenticity, and confidentiality. In *AsiaCCS*, pages 461–468. ACM, 2017.

[33] J. Braun, J. A. Buchmann, C. Mullan, and A. Wiesmaier. Long term confidentiality: a survey. *Des. Codes Cryptography*, 71(3):459–478, 2014.

[34] D. Catalano and D. Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *ACM Conference on Computer and Communications Security*, pages 1518–1529. ACM, 2015.

[35] D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 538–555. Springer, 2014.

[36] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 680–699. Springer, 2013.

[37] D. Catalano, D. Fiore, and L. Nizzardo. Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In *CRYPTO (2)*, volume 9216 of *Lecture Notes in Computer Science*, pages 254–274. Springer, 2015.

[38] D. Catalano, D. Fiore, and L. Nizzardo. Homomorphic signatures with sublinear public keys via asymmetric programmable hash functions. *Des. Codes Cryptography*, 86(10):2197–2246, 2018.

[39] D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. *IACR Cryptology ePrint Archive*, 2011:696, 2011.

[40] D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 680–696. Springer, 2012.

[41] D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 371–389. Springer, 2014.

[42] D. Catalano, A. Marcedone, and O. Puglisi. Linearly homomorphic structure preserving signatures: New methodologies and applications. *IACR Cryptology ePrint Archive*, 2013:801, 2013.

[43] D. Catalano, A. Marcedone, and O. Puglisi. Authenticating computation on groups: New homomorphic primitives and applications. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 193–212. Springer, 2014.

[44] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2-3):141–167, 2010.

[45] B. Chevallier-Mames, J. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing. In *CARDIS*, volume 6035 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2010.

[46] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2013.

[47] K. Chung, Y. T. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010.

[48] J. Coron, M. S. Lee, T. Lepoint, and M. Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 607–628. Springer, 2016.

[49] J. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2013.

[50] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society, 2015.

[51] R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[52] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[53] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[54] D. Demirel, M. Henning, J. van de Graaf, P. Y. A. Ryan, and J. A. Buchmann. Prêt à voter providing everlasting privacy. In *VOTE-ID*, volume 7985 of *Lecture Notes in Computer Science*, pages 156–175. Springer, 2013.

[55] Y. Desmedt. Computer security by redefining what a computer is. In *NSPW*, pages 160–166. ACM, 1993.

[56] K. Elkhiyaoui, M. Önen, M. Azraoui, and R. Molva. Efficient techniques for publicly verifiable delegation of computation. In *AsiaCCS*, pages 119–128. ACM, 2016.

[57] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

[58] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *ACM Conference on Computer and Communications Security*, pages 844–855. ACM, 2014.

[59] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT (2)*, volume 10032 of *Lecture Notes in Computer Science*, pages 499–530, 2016.

[60] D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 697–714. Springer, 2012.

[61] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[62] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[63] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[64] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 142–160. Springer, 2010.

[65] R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2013.

[66] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[67] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.

[68] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[69] S. Goldwasser, S. Micali, and A. C. Yao. Strong signature schemes. In *STOC*, pages 431–439. ACM, 1983.

[70] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, pages 469–477. ACM, 2015.

[71] S. D. Gordon, J. Katz, F. Liu, E. Shi, and H. Zhou. Multi-client verifiable computation with stronger security guarantees. In *TCC (2)*, volume 9015 of *Lecture Notes in Computer Science*, pages 144–168. Springer, 2015.

[72] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[73] R. Johnson, D. Molnar, D. X. Song, and D. A. Wagner. Homomorphic signature schemes. In *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2002.

[74] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. TRUESET: faster verifiable set computations. In *USENIX Security Symposium*, pages 765–780. USENIX Association, 2014.

[75] J. Lai, R. H. Deng, H. Pang, and J. Weng. Verifiable computation on outsourced encrypted data. In *ESORICS (1)*, volume 8712 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2014.

[76] R. W. F. Lai, R. K. H. Tai, H. W. H. Wong, and S. S. M. Chow. Multi-key homomorphic signatures unforgeable under insider corruption. In *ASIACRYPT (2)*, volume 11273 of *Lecture Notes in Computer Science*, pages 465–492. Springer, 2018.

[77] H. T. Lee and J. H. Seo. Security analysis of multilinear maps over the integers. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 224–240. Springer, 2014.

[78] B. Libert, T. Peters, M. Joye, and M. Yung. Linearly homomorphic structure-preserving signatures and their applications. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 289–307. Springer, 2013.

[79] B. Libert, S. C. Ramanna, and M. Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *ICALP*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[80] B. Libert and M. Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 499–517. Springer, 2010.

[81] T. Matsuda and G. Hanaoka. Chosen ciphertext security via UCE. In *Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 56–76. Springer, 2014.

[82] E. Miles, A. Sahai, and M. Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 629–658. Springer, 2016.

[83] T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, 2006.

[84] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[85] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 222–242. Springer, 2013.

[86] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[87] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*,

volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[88] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[89] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[90] L. Schabhüser, D. Butin, D. Demirel, and J. Buchmann. Function-dependent commitments for verifiable multi-party computation. In *ISC*, volume 11060 of *Lecture Notes in Computer Science*, pages 289–307. Springer, 2018.

[91] L. Schabhüser, D. Demirel, and J. A. Buchmann. An unconditionally hiding auditing procedure for computations over distributed data. In *CNS*, pages 552–560. IEEE, 2016.

[92] B. Schoenmakers and M. Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *ACNS*, volume 9092 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[93] B. Schoenmakers, M. Veeningen, and N. de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In *ACNS*, volume 9696 of *Lecture Notes in Computer Science*, pages 346–366. Springer, 2016.

[94] S. T. V. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys*, pages 71–84. ACM, 2013.

[95] S. T. V. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*. The Internet Society, 2012.

[96] S. T. V. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium*, pages 253–268. USENIX Association, 2012.

[97] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[98] P. Struck, L. Schabhüser, D. Demirel, and J. A. Buchmann. Linearly homomorphic authenticated encryption with provable correctness and public

verifiability. In *C2SI*, volume 10194 of *Lecture Notes in Computer Science*, pages 142–160. Springer, 2017.

[99] Y. Sun, Y. Yu, X. Li, K. Zhang, H. Qian, and Y. Zhou. Batch verifiable computation with public verifiability for outsourcing polynomials and matrix computations. In *ACISP (1)*, volume 9722 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2016.

[100] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.

[101] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *HotCloud*. USENIX Association, 2012.

[102] V. Vu, S. T. V. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 223–237. IEEE Computer Society, 2013.

[103] G. Xu, G. T. Amariucai, and Y. Guan. Verifiable computation with reduced informational costs and computational costs. In *ESORICS (1)*, volume 8712 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2014.

[104] L. F. Zhang and R. Safavi-Naini. Generalized homomorphic macs with efficient verification. In *AsiaPKC@AsiaCCS*, pages 3–12. ACM, 2014.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, März 2019 _____

Wissenschaftlicher Werdegang

März 2015 - heute

Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt

Oktober 2011 - Dezember 2014

Studium im Studiengang „Mathematik mit Nebenfach Wirtschaftswissenschaften“ (Master of Science) an der Johannes Gutenberg Universität Mainz

April 2008 - September 2018 & April 2009 - Juli 2011

Studium im Studiengang „Mathematik mit Nebenfach Wirtschaftswissenschaften“ (Bachelor of Science) an der Johannes Gutenberg Universität Mainz